

AD-A069 241

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 12/1
INVESTIGATION OF INVERSE VANDERMONDE MATRIX CALCULATION FOR LIN--ETC(U)
MAR 79 D P SEYLER

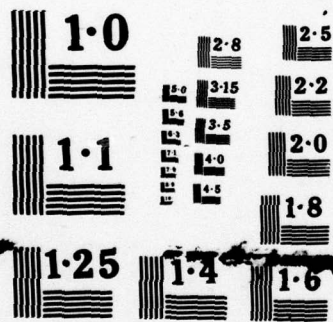
UNCLASSIFIED

AFIT/66C/EE/79-2

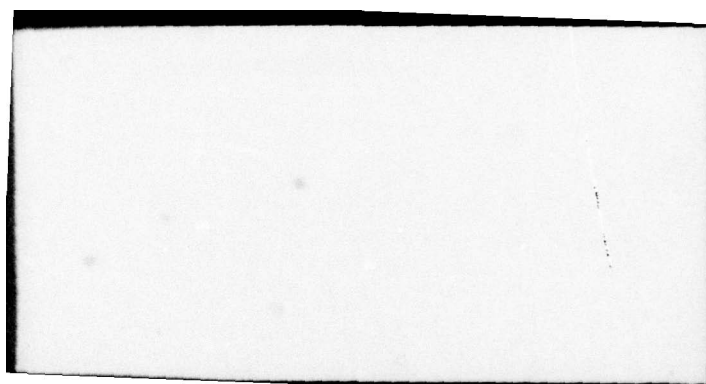
NL

1 OF 2
AD
A039241





NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART



DISCLAIMER NOTICE

**THIS DOCUMENT IS BEST QUALITY
PRACTICABLE. THE COPY FURNISHED
TO DDC CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

①

⑥

INVESTIGATION OF INVERSE VANDERMONDE
MATRIX CALCULATION FOR LINEAR
SYSTEM APPLICATIONS.

⑨ Master's THESIS

⑭

AFIT/GGC/EE/79-2

⑩

Donald P. Seyler
2nd Lt USAF

DDC
RECEIVED
JUN 1 1979
A

⑪ Mar 79

⑫ 178 p.

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

012225

B

79 05 29 109

INVESTIGATION OF INVERSE VANDERMONDE
MATRIX CALCULATION FOR LINEAR
SYSTEM APPLICATIONS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air Training Command
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Donald P. Seyler, B.S.E.E.

2nd Lt

USAF

Graduate Electrical Engineering

March 1979

| | |
|---------------------------------|---|
| ABSTRACTED BY | |
| RTIS | WORK DONE <input checked="" type="checkbox"/> |
| FTC | DATA ACQUIRED <input type="checkbox"/> |
| ORIGINATOR | <input type="checkbox"/> |
| NOTIFICATION | |
| BY | |
| STANDARDIZATION SYMBOLS CHECKED | |
| DATE | |
| A | 23 JYL |

Approved for public release; distribution unlimited.

Preface

This thesis is intended as a source document for further research in system identification processes. Specifically, this research concentrated on efficiently finding the inverse of confluent (i.e., repeated eigenvalue) Vandermonde matrices, which is an important part of the component matrix approach to system identification. However, in order to provide a broader understanding of system identification in general, and the component matrix approach in particular, the background and theory sections are more comprehensive than might otherwise be needed. Several supplementary mathematical developments and some example problems are included to aid this broad understanding, but they are placed in appendices to preserve the continuity of the text, and referenced as appropriate. A list of special symbols and abbreviations is also included, to give the reader a ready reference for those that may be unfamiliar.

I wish to give special thanks to Maj. J. Gary Reid, my thesis advisor, for his support and encouragement in this endeavor. Professor Charles W. Richard and Dr. Peter S. Maybeck, who read the manuscript, gave many helpful suggestions. For their patient consultation regarding the computer programming and example problems used in the course of the research, I wish to thank Maj. Edward Reeves and Lt. Paul Vergez. I am also deeply indebted to a close personal friend of mine, Rev. Roy Dorsett, for his understanding support

throughout the entire project. There are many others like him who "cheered from the sidelines," but are too numerous to mention. Finally I want to thank Mrs. Evelyn Shaw for her perseverance in the final days while she was typing the final draft.

Donald P. Seyler

Contents

| | Page |
|--|------|
| Preface | ii |
| List of Figures | vi |
| List of Tables | vii |
| Symbols and Abbreviations | viii |
| Abstract | ix |
| I. Introduction | 1 |
| Motivational Background | 2 |
| System Identification Process Background | 6 |
| II. Parameter Sensitivity Calculations | 9 |
| Coupled Linear Differential Equations | |
| Method | 11 |
| Component Matrix Method | 13 |
| III. Vandermonde Matrix Inversion | 19 |
| Potential Problem Sources | 19 |
| Complex Eigenvalues | 20 |
| Repeated Eigenvalues | 20 |
| Nearly Identical or Nearly Zero | |
| Eigenvalues | 21 |
| Eigenvalues With Large Magnitude | |
| Differences | 21 |
| Csáki's Generalized Algorithm | 21 |
| Parallel Architecture Processors and | |
| Csáki's Generalized Algorithm | 25 |
| IV. Tests Run on the Coded Version of Csáki's | |
| Algorithm | 30 |
| Types of Tests | 31 |
| Accuracy Tests | 31 |
| V. Results of Tests | 36 |
| Computation Time | 36 |
| Accuracy as a Function of System Order | 43 |
| Accuracy as a Function of $ \lambda_x - \lambda_y _{\min}$ | 45 |

Contents

| | Page |
|---|------|
| Accuracy as a Function of Condition Index | 45 |
| Accuracy as a Function of Increasing Condition Index Caused by an Eigenvalue Approaching Zero | 47 |
| VI. Conclusions and Recommendations | 50 |
| Conclusions | 50 |
| Recommendations | 51 |
| Bibliography | 57 |
| Vita | 59 |
| Appendix A: Flowcharts and Explanation of subroutine VANINV | A-1 |
| Appendix B: Program Source Listings | B-1 |
| Appendix C: Supplementary Mathematical Developments | C-1 |
| Appendix D: Example Problems | D-1 |

List of Figures

| <u>Figure</u> | | <u>Page</u> |
|---------------|--|-------------|
| 1 | Conceptual model of an adaptive feedback control system | 5 |
| 2 | Processing time comparison | 27 |
| | a) Serial processor | |
| | b) Parallel processor | |
| 3 | Parallel computation structure of component matrix method of system identification | 28 |
| 4 | Computation time as a function of system order (30 iterations with real eigenvalues) | 42 |
| 5 | Computation accuracy as a function of system order (real eigenvalues) | 44 |
| 6 | Computation accuracy as a function of two closest eigenvalues ($ \lambda_x - \lambda_y _{\min}$) | 46 |
| 7 | Computation accuracy as a function of increasing condition index caused by an eigenvalue approaching zero | 48 |
| 8 | Example of VANINV calculation applied to a fourth order system with a repeated complex conjugate pair of eigenvalues | 53 |
| 9 | Example of VANINV calculations applied to a fourth order system with like eigenvalues | 55 |

List of Tables

| <u>Table</u> | | <u>Page</u> |
|--------------|------------------------|-------------|
| I | Test Systems | 37 |
| II | Test Results | 40 |

Symbols and Abbreviations

CLDE = Coupled Linear Differential Equation

CM = Component Matrix

n = System order

p = Number of system parameters

P/O = "Part Of"

q = Number of system outputs

r = Number of system control inputs

STM = State Transition Matrix

w.r.t. = "With Respect To"

$\begin{matrix} \overline{x} \\ \underline{\overline{A}} \end{matrix}$ = Augmented vector and augmented matrix

1. Lower case denotes a vector

2. Upper case denotes a matrix

3. Underscore denotes "vector" or "matrix" as appropriate; used in addition to 1 and 2 if confusion with scalars is likely

4. Overscore denotes "augmented" in either case

$\begin{matrix} \overline{x} \\ \underline{\overline{A}} \end{matrix} (i) (\cdot) =$ Partial derivatives of the system state and system matrix w.r.t. the i 'th system parameter, θ_i

θ_i = i 'th parameter component of a state space system

Abstract

Efficient inversion of a $2n \times 2n$ Vandermonde matrix is a key requirement of a new algorithm developed by J. Gary Reid (Ref 10, 11) for parameter identification in linear time-invariant systems. It has features very desirable for application to large systems with many unknown parameters, such as adaptive flight control systems.

A generalized algorithm for inverting the Vandermonde matrix was proposed by F. G. Csáki (Ref 1). It was chosen for study because its structure parallels that of Reid's algorithm. Csáki's algorithm was coded into a computer subroutine called "VANINV," and 43 eigensystems were used to test its computational accuracy and efficiency. Four potential problem areas tested were: (1) large system orders, (2) eigenvalues near each other, (3) eigenvalues near zero, and (4) eigenvalues with large magnitude differences. For a comparison, the Vandermonde matrix for each system was also inverted using routines from the International Mathematical & Statistical Library (IMSL).

Test results indicated that VANINV is not quite as fast or as accurate as the IMSL routines. However, the tests were limited to systems with real distinct eigenvalues because the IMSL routines cannot handle other types. VANINV in its present form can handle systems having any combination of complex and/or repeated eigenvalues. Therefore, recommendations for further research and several possible means of improving VANINV are outlined.

INVESTIGATION OF INVERSE VANDERMONDE MATRIX CALCULATION FOR LINEAR SYSTEM APPLICATIONS

I Introduction

Fundamental quantities needed for iterative computation of a quasi-linear estimate of unknown parameters in a linear dynamic system are the "parameter sensitivity variables." Finding these variables is a basic part of a "system identification process," as is shown in the background subsection on system identification.

Section II develops the process for finding the parameter sensitivity variables and shows that a basic part of the process is the inversion of a $2n \times 2n$ Vandermonde matrix. The development of an efficient computer program to invert the Vandermonde matrix, the major emphasis of this thesis effort, is presented in Section III. Section IV explains some test procedures used for determining the accuracy and efficiency of the program. The concluding two sections give the results of the research and recommendations for further research.

As an aid in understanding the concepts and processes presented in the theory sections of the thesis, several example problems and supplementary mathematical developments are worked out. However, because of their length they are placed in appendices and referenced as appropriate. The computer program developed to invert the Vandermonde

matrix and the programs used to test its computational speed and accuracy are also placed in appendices.

Motivational Background

If all systems were linear and time-invariant, the linear system model (noise-free):

$$\dot{x} = Ax + Bu \quad (1a)$$

$$y = Cx + Du \quad (1b)$$

would suffice. However, control systems for other than very simple applications are not linear or time-invariant: depending on operating conditions,¹ the control laws must change to maintain adequate system control. For example, the Wright Brothers didn't have to concern themselves with large changes in operating conditions. Their first powered flight, 120 feet in 12 seconds (Ref 2:Vol 29:557; 14:Vol 7:388), was not even as fast as a man can run, and didn't last long enough for the environment to change much.

On the other hand, engineers designing flight control systems for modern high performance aircraft do need to concern themselves with large changes in operating conditions. As aircraft flight envelopes are extended over broader operating ranges (for example, a high performance

¹In this study, "operating conditions" is used to encompass both system configuration and environment. Each has definite effects on system response, and may also affect the other. For convenience however, the term "operating conditions" is used to refer to both when it is not necessary to distinguish between them.

aircraft may be required to have a Vertical or Short Take-off and Landing (V/STOL) capability), more and alternate loops must be added to the flight control system to handle all the various operating conditions. The increased requirement for control system flexibility often results in an overall system that is nonlinear and/or time-varying, and to maintain adequate control, some means of adjusting the control system parameters is necessary. This capacity to change the modes of the control system according to the aircraft flight operating conditions is the essence of an "adaptive" flight control system. Of course every time the operating conditions change, the control system variables must change accordingly. Thus the control problem becomes one of identifying the system parameters (the process is called "system identification") in order to determine the proper system model and match an appropriate control law to it.

The flight control system design engineer could try to think of a collection of representative types of operating conditions which would, hopefully, cover the continuum of possible operating conditions. A control loop could then be designed to handle each different representative operating condition. That would require an a priori identification of the system model and parameter values and a corresponding collection of hardware to implement the design. Then some type of master controller would be needed to choose which model and parameter values to use

for the given operating conditions.

In terms of an entirely analog flight control system, the increased flexibility required to handle the adaptive part of the control function results in a confusing mass of hardware and interconnecting links. Also, each new piece of hardware poses a possible reliability problem. These are some of the reasons for the trend toward digital flight control systems. Instead of having a discrete piece of hardware for each required control function, a digital controller need be simply reprogrammed to accommodate the new control system parameters. These parameters provide the relation between the system inputs, states, and outputs for the control variable calculations.

The overall control system can be visualized as shown in Figure 1. Upon starting the system, some nominal system model and control are used. However, as shown, the model must change to account for significant variations in the system operating conditions. Referring back to the high performance aircraft mentioned before, an example of a change in the system configuration is the change from a vertical takeoff to "conventional" horizontal flight. An example of an environmental change affecting the system model is the change in air density with altitude (eg: the air at 40,000 feet is much less dense than at sea level, so the control surface actuator gains must be changed accordingly to maintain the same performance relative to pilot inputs). The digital control system can be given

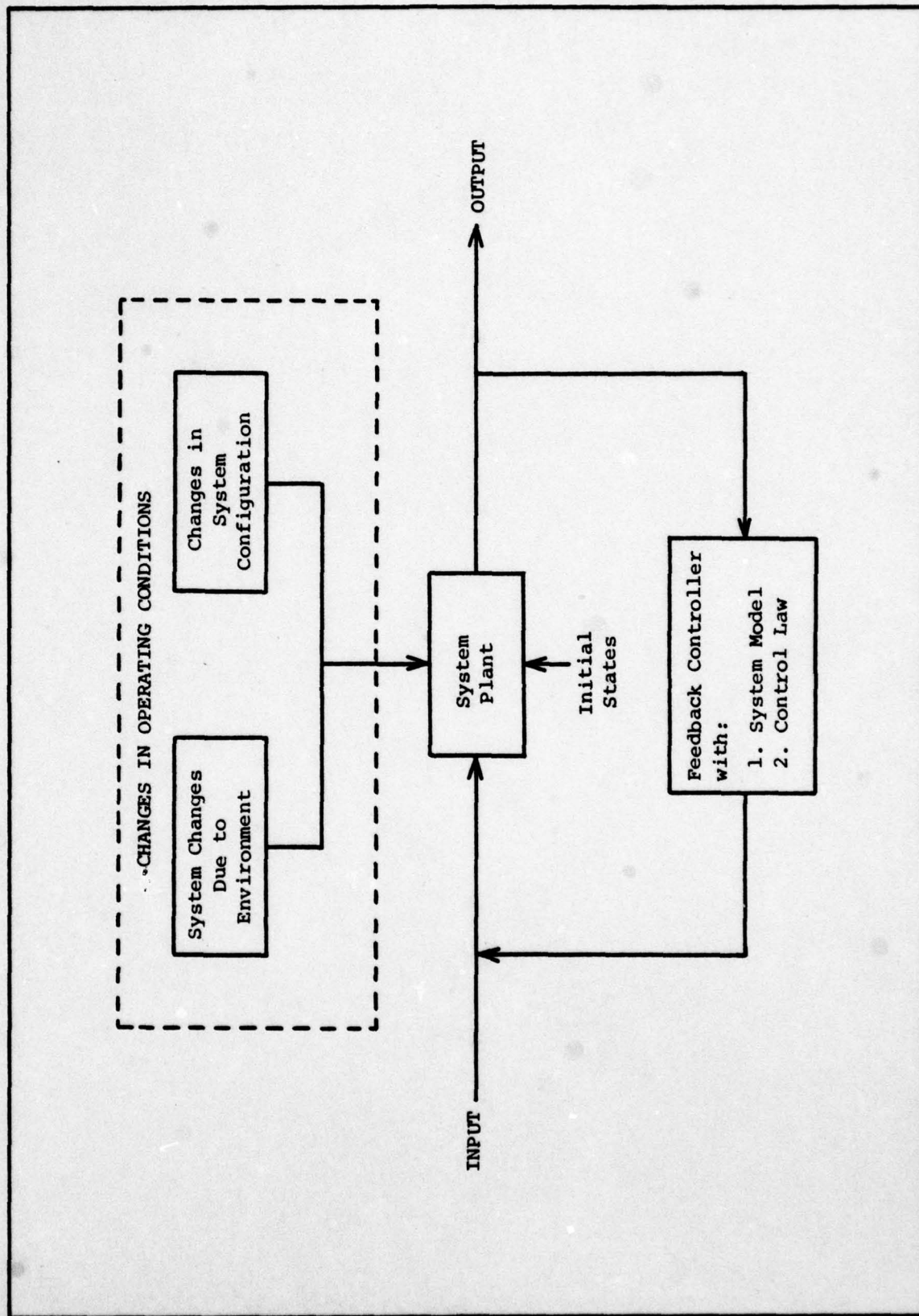


Figure 1. Conceptual model of an adaptive feedback control system.

the task of identifying the present system parameters, updating the system model with them, and then determining what adaptations must be made in the control to achieve the desired overall performance. The first two steps are what distinguish an adaptive control system from a fixed control system.

System Identification Process Background

A conceptually simple method for the system identification process is the use of a first order Taylor series approximation to update the system model according to:

$$y(t) \approx \hat{y}(t) \Big|_{\substack{\text{a priori} \\ \text{model}}} + \frac{\partial y(t)}{\partial \theta(t)} \Delta \theta \quad (2)$$

where:

$y(t)$ = physically measured system output

$\hat{y}(t)$ = calculated output based on an a priori model (or in an iterative process, the last updated model)

$\frac{\partial y(t)}{\partial \theta(t)}$ = system output "sensitivity" with respect to the system parameters

$\Delta \theta$ = change in the system parameters from those of the a priori model (or the change since the last model update, for an iterative process)

Notice that since $\Delta y = \frac{\partial y}{\partial \theta} \Delta \theta$, Eq (2) is equivalent to

$$y(t) = \hat{y}(t) + \Delta y(t) \quad (3)$$

which appeals to the intuitive relation "new output equals old output plus the change in the output."

It is seen, then, that finding these sensitivity variables is a significant part of the system identification process. Therefore, real-time adaptive control applications, such as the adaptive flight control system mentioned earlier, require an efficient means of computing the sensitivity variables.

However, even when using minimal order sensitivity models, the computational requirements can be quite large for even modest size systems. For example, the complete solution of a sensitivity system of order n , with p unknown parameters, may require the solution of $n(p+1)$ coupled linear differential equations (Ref 8:124]. The basis for this number is presented in Section II.

An alternate method has been proposed by J. G. Reid in his dissertation (Ref 11:Section III] which may have significant computational advantages because it avoids the need to solve a collection of coupled linear differential equations (CLDE). This method is explained in more detail in Section II.

However, this alternate method also involves the inversion of a $2n \times 2n$ Vandermonde matrix, as is shown in Section II. (Vandermonde matrices are defined in Appendix C, Eqs (C2-2 through C2-6)). This process can be computationally quite costly (Ref 5:817). Therefore, a critical part of the proposed alternate method is the efficient inversion of the Vandermonde matrix. Thus the effort of this investigation

concentrates on implementing and evaluating a proposed method (Ref 1:154-157) of efficiently computing the inverse Vandermonde matrix. The inversion method used forms polynomials from the system eigenvalues and uses the polynomials to determine the elements of the inverse Vandermonde matrix. Section III amplifies this method and shows its parallel computation structure. The significance of this parallel structure is explained in terms of its potential application to on-line flight controllers. Section IV explains the tests used to evaluate the computation time and accuracy of the computer subroutine written to implement the method. The subroutine was called VANINV. Sections V and VI give the test results and resulting conclusions and resulting recommendations for further research.

II Parameter Sensitivity Calculations

Section I showed that calculation of the parameter sensitivities is an essential part of the system identification process. In this section, a theoretical comparison is made between the coupled linear differential equations (CLDE) approach to finding parameter sensitivities and the alternate component method (CM) approach proposed by Reid. Since the CLDE approach is already developed and documented (eg: Ref 13), it is merely outlined here to serve as a reference base for the CM approach.

Although the CLDE method is more straightforward, it has several drawbacks (Ref 8:124). Perhaps the most obvious difficulty is the computational burden. As is illustrated later in this section, the solution of the sensitivity system requires solving as many as $n(p+1)$ coupled linear differential equations, where n is the system dimension and p is the number of parameters whose sensitivities are to be found.¹ Attempts to reduce this computational burden with low-order sensitivity models have resulted in some other problems, such as: (1) requirement for special forms of the plant matrix, (2) numerical sensitivity of calculations performed with the reduced

¹This task grows increasingly formidable as n and p increase. Each new variable not only adds another CLDE, it may contribute to the complexity of any or all of the other equations.

order models, and (3) difficulty of obtaining the original state variable sensitivities from the calculated low-order sensitivities. In addition, physical insight is impaired when using the transformed coordinate systems resulting from the low-order models.

Among the advantages of the CM method are: (1) potential reduction in computational burden, (2) retention of physical system insight (the modes are visible throughout the calculation procedure), and essentially "free" availability of the eigenvalue sensitivities themselves (Ref 11:79-80). The entire set of parameter sensitivities may be found from, at most, $2nr$ "quadrature integrals" (where n is the system dimension and r is the number of control inputs), plus some matrix operations (Ref 8:133). The last part of this section shows that the CM method requires the inversion of a $2n \times 2n$ Vandermonde matrix. Since this inversion is potentially very costly, Section III is devoted to investigating the possible method that was referenced in the introduction for efficiently computing the inverse. At this point, some system definitions are needed.

Most control systems other than very simple ones are not time-invariant. However, assuming that the variations are slow w.r.t. the identification process (as would be the case for flight control systems), the system may be considered to be pseudo time-invariant (or "quasi-static"). Thus, for this discussion, the linear time-invariant state system is used:

$$\dot{x}(t) = A(\theta) x(t) + B(\theta)u(t) \quad (4a)$$

$$y(t) = C(\theta) x(t) + D(\theta)u(t) \quad (4b)$$

The relation of the initial conditions to the parameter sensitivities is defined by the vector

$$x(0) = \phi(\theta) \quad (5)$$

This shows that the system matrices and the initial states are a function of the "parameter vector," θ , of which there are p elements, designated θ_i , $i = 1, 2, \dots p$. The nominal value, θ_0 , of the parameter vector is used for all evaluations. Furthermore, it is assumed that A , B , C , D and ϕ are real, bounded and continuously differentiable w.r.t. the parameter component θ_i at its nominal value.

The "state sensitivities" of this system are defined as the partial derivative of the system state vector w.r.t. the parameter components θ_i , and are written as

$$\underline{x}_{(i)}(t) = \left. \frac{\partial x(t; \theta)}{\partial \theta_i} \right|_{\theta = \theta_0} \quad (6)$$

Coupled Linear Differential Equations Method

With the assumptions made in the introduction for this section, the system parameter sensitivities may be found by augmenting the original system with the desired state sensitivities (from Eq (6)) as follows (Ref 8:124):

$$\bar{\underline{x}} \equiv \begin{bmatrix} \underline{x} \\ \underline{x}_{(1)} \\ \vdots \\ \underline{x}_{(p)} \end{bmatrix} \quad \bar{\underline{y}} \equiv \begin{bmatrix} \underline{y} \\ \underline{y}_{(1)} \\ \vdots \\ \underline{y}_{(p)} \end{bmatrix} \quad \bar{\underline{\phi}} \equiv \begin{bmatrix} \underline{\phi} \\ \underline{\phi}_{(1)} \\ \vdots \\ \underline{\phi}_{(p)} \end{bmatrix} \quad (7)$$

$n(p+1) \times 1 \quad q(p+1) \times 1 \quad n(p+1) \times 1$

$$\bar{\underline{A}} \equiv \begin{bmatrix} \underline{A} & 0 & \dots & 0 \\ \underline{A}_{(1)} & \underline{A} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \underline{A}_{(p)} & 0 & \dots & \underline{A} \end{bmatrix} \quad \bar{\underline{B}} \equiv \begin{bmatrix} \underline{B} \\ \underline{B}_{(1)} \\ \vdots \\ \underline{B}_{(p)} \end{bmatrix} \quad (8)$$

$n(p+1) \times n(p+1) \quad n(p+1) \times r$

$$\bar{\underline{C}} \equiv \begin{bmatrix} \underline{C} \\ \underline{C}_{(1)} \\ \vdots \\ \underline{C}_{(p)} \end{bmatrix} \quad \bar{\underline{D}} \equiv \begin{bmatrix} \underline{D} \\ \underline{D}_{(1)} \\ \vdots \\ \underline{D}_{(p)} \end{bmatrix} \quad (9)$$

$n(p+1) \times q \quad n(p+1) \times r$

where q is the system output dimension and r is the system input (control) dimension. This augmentation gives a "sensitivity system" which contains all the original states plus the desired parameter sensitivities expressed as states. From Eq (7) it is evident that the complete solution of this sensitivity system involves solving up to $n(p+1)$ coupled linear differential equations.

Component Matrix Method

The CM method of finding the parameter sensitivities seems at first to be much more indirect than the CLDE method. However, after all the development of the theory is complete, it turns out that some steps can be eliminated because they serve only as convenient dividing points for the problem when solved manually. In a computer solution process, once the inverse Vandermonde matrix is found, all the sensitivities of the parameters and the original system can be determined using the elements of the matrix.

Eq (4b), the system output equation, has a time domain solution (Ref 12:373)

$$y(t) = Ce^{A(t-t_0)}x(t_0) + \int_{t_0}^t e^{A(t-\tau)}Bu(\tau)d\tau + Du(t) \quad (10)$$

For simplicity of discussion, the initial time, t_0 , and the feed forward matrix, D , may be set to zero. Since the system is time-invariant, the output matrix, C , can be taken outside of the integral. Thus Eq (1) becomes

$$y(t) = Ce^{At}x(0) + C \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau \quad (11)$$

The state transition matrix (STM), e^{At} , is not a function of the variable of integration, so it may be removed from the integral. Therefore

$$y(t) = Ce^{At}x(0) + Ce^{At} \int_0^t e^{-A\tau}Bu(\tau)d\tau \quad (12)$$

Notice that the integral is now only a function of one variable and is therefore simple to compute. Such single

variable integrals are called "quadrature integrals." Notice that there are a total of nr quadrature integrals since $e^{-A\tau}$ is an $n \times n$ matrix and B is an $n \times r$ matrix.

Eq (2) established the need to find the system output parameter sensitivities, so differentiating Eq (12) w.r.t. the parameter vector, θ , yields

$$\begin{aligned} \frac{\partial y}{\partial \theta_i} = & C e^{At} \frac{\partial x(0)}{\partial \theta_i} + C \frac{\partial}{\partial \theta_i} e^{At} x(0) + \frac{\partial}{\partial \theta_i} C e^{At} x(0) \\ & + C \int_0^t e^{A\tau} \frac{\partial}{\partial \theta_i} B u(\tau) d\tau \\ & + C \int_0^t \frac{\partial}{\partial \theta_i} e^{A\tau} B u(\tau) d\tau \\ & + \frac{\partial}{\partial \theta_i} C \int_0^t e^{A\tau} B u(\tau) d\tau \end{aligned} \quad (13)$$

Note that none of the parameter sensitivity derivatives in Eq (13) are functions of time except that of the state transition matrix:

$$e_{(i)}^{At} \equiv \frac{\partial}{\partial \theta_i} e^{At} \quad (14)$$

The sensitivity derivative of the state transition matrix is potentially very costly to obtain because, in the form shown in Eq (14), it must be reevaluated for each desired point in time.

It can be shown (see Appendix C, Section 1), that the sensitivity derivative of the original state transition matrix, $e_{(i)}^{At}$, is contained in the evaluation of the state transition matrix of the augmented sensitivity system, i.e:

$$\begin{bmatrix} e^{At} & | & 0 \\ \hline & & \\ e_{(i)} & | & e^{At} \end{bmatrix}_{2n \times 2n} = e^{\begin{bmatrix} A & | & 0 \\ \hline & & \\ A_{(i)} & | & A \end{bmatrix} t} \quad (15)$$

By an application of the Cayley-Hamilton theorem (see Appendix C, section 2) the state transition matrix of the augmented sensitivity system is found to be a summation of products of the remainder polynomial coefficients and powers of the sensitivity system matrix:

$$e^{\begin{bmatrix} A & | & 0 \\ \hline & & \\ A_{(j)} & | & A \end{bmatrix} t} = \sum_{j=1}^{2n} \begin{bmatrix} A & | & 0 \\ \hline & & \\ A_{(i)} & | & A \end{bmatrix}^{j-1} a_j(t) \quad (16)$$

However

$$\begin{bmatrix} A & | & 0 \\ \hline & & \\ A_{(i)} & | & A \end{bmatrix}^j = \begin{bmatrix} A^j & | & 0 \\ \hline & & \\ (A^j)_{(i)} & | & A^j \end{bmatrix} \quad (17)$$

that is: the power of a matrix can be brought inside and applied to each element (Ref 11:64; 3:241-248). Therefore applying Eqs. (15) and (17) to Eq (16) results in

$$\begin{bmatrix} e^{At} & | & 0 \\ \hline & & \\ e_{(i)} & | & e^{At} \end{bmatrix} = e^{\begin{bmatrix} A & | & 0 \\ \hline & & \\ A_{(i)} & | & A \end{bmatrix} t} = \sum_{j=1}^{2n} \begin{bmatrix} A^{j-1} & | & 0 \\ \hline & & \\ A_{(i)}^{j-1} & | & A^{j-1} \end{bmatrix} a_j(t) \quad (18)$$

Eq (18) shows that the scalar multiplication and summation operations can be applied separately to each section.

of the partitioned state transition matrix of the augmented sensitivity system. So the partial derivative of the original state transition matrix, which is needed to find the parameter sensitivities according to Eq (13), can be evaluated simply as

$$e_{(i)}^{At} = \frac{\partial}{\partial \theta_i} e^{At} = \sum_{j=1}^{2n} A_{(i)}^{j-1} a_j(t) \quad (19)$$

At this point, it is still necessary to find the partial derivative of $2n$ powers of the original system matrix, A , and solve $2n$ differential equations to find the remainder polynomial coefficients, $a_j(t)$. The sensitivity problem appears to have become even more complicated.

However, Reid shows (Ref 11:70) that the partial derivative of the system state transition matrix can be found from a summation of products of partial derivatives of the system "component matrices" and the system modes:

$$e_{(i)}^{At} = \sum_{j=1}^n (z_{j,o})_{(i)} e^{\lambda_j t} + \sum_{j=1}^n (z_{j,o})_{(i)} t(\lambda_j) e^{\lambda_j t} \quad (20)$$

Eq (20) is a simplification of Reid's formulation and applies for the case of distinct eigenvalues. It is used here for clarity.

Eq (20) can be extended to the augmented system:

$$e^{\begin{bmatrix} A & | & 0 \\ \hline A_{(i)} & | & A \end{bmatrix} t} = \sum_{j=1}^n \begin{bmatrix} z_{j,o} & | & 0 \\ \hline (z_{j,o})_{(i)} & | & z_{j,o} \end{bmatrix} e^{\lambda_j t} + \sum_{j=1}^n \begin{bmatrix} 0 & | & 0 \\ \hline z_{j,o} & | & 0 \end{bmatrix} t(\lambda_j)_{(i)} e^{\lambda_j t} \quad (21)$$

It is evident from Eq (21) that for every mode, $e^{\lambda_j t}$, in the original system, there is an additional mode, $t e^{\lambda_j t}$, in the augmented system. However, the additional modes occur only in the sensitivity derivative of the STM. Thus the STM of the original matrix is preserved by the CM method. Only the original modes appear in the STM for the original system, e.g.:

$$e^{At} = \sum_{j=1}^n (z_{j,o}) e^{\lambda_j t} \quad (22)$$

Thus the CM method is seen to have a physical insight advantage in comparison to the CLDE method: the modes of the system are clearly visible throughout the CM solution process, whereas usually they are not in the CLDE process. But there is now a total of $2n$ modes, so the Vandermonde matrix which must be inverted to find the component matrices for the sensitivity system is $2n \times 2n$. (See

Appendix C, Section 3 for a discussion of how the component matrices are formed.) And, a total of $2nr$ quadrature integrals must be found.

This section theoretically compared the CLDE method of computing system parameter sensitivities to the alternate CM method proposed by Reid. It showed that when the state system is augmented to include the parameter sensitivities, the CLDE method requires the solution of up to $n(p+1)$ coupled linear differential equations. To reduce this computational burden, reduced order models are sometimes used, but the required coordinate transformations often impair physical insight into the system and make it difficult to obtain the original system sensitivities from the reduced order sensitivities. The alternate procedure, the CM method, was shown to retain the physical significance of the system variables throughout the calculation process, and to exchange the formidable task of solving $n(p+1)$ coupled linear differential equations for $2nr$ much simpler quadrature integrals, plus some matrix operations. However, the potentially costly process of inverting a $2n \times 2n$ Vandermonde matrix was also shown to be part of the CM method. Thus Section III deals with the subject of efficiently inverting the Vandermonde matrix.

III Vandermonde Matrix Inversion

Preceding sections established the need to calculate parameter sensitivities as part of a system identification process. Two methods for finding those parameter sensitivities were compared: the CLDE method and the CM method. The latter was chosen for further study based on theoretical comparisons in Section II. However, a key issue associated with this method is the need to invert a $2n \times 2n$ Vandermonde matrix. This section addresses a number of problems associated with inverting the Vandermonde matrix, and investigates a generalized method of Vandermonde inversion proposed by F. G. Csáki (Ref 1:154-157). A special feature of Csáki's generalized algorithm is also discussed. That feature is that the algorithm can be implemented on a parallel architecture processor, which for a real-time flight control system in particular, is significant.

Potential Problem Sources

There are basically five sources of problems associated with computing the inverse of the Vandermonde matrix, all of them related to the characteristics of the eigenvalues.

They are:

- (1) Complex eigenvalues
- (2) Repeated eigenvalues
- (3) Eigenvalues close to each other

- (4) Eigenvalues close to the origin
- (5) Combinations of very large and very small eigenvalues

Each of these potential problems is addressed in turn.

Complex Eigenvalues. Vandermonde matrices formed from complex variables may be handled three ways: (1) conversion of the complex matrix to a real one (Ref 9: Problem 6.8), (2) use of two separate sets of real variables for all calculations--one set for the real parts and the other for the imaginary parts of the complex variables, and (3) use of complex variables directly. Investigation of Csáki's generalized algorithm was desirable because of its parallel implementation capability, as is discussed later in this section. The algorithm has an inherent capability for handling complex eigenvalues, so conversion of the complex Vandermonde to a real one was unnecessary. Few control systems have entirely complex eigenvalues, so with this in mind the second method of handling complex variables was chosen. In this way, tests for complexity can be performed, and for calculations involving only real quantities, the complex part of the calculations can be omitted to save computation time.

Repeated Eigenvalues. For systems with repeated eigenvalues, the derivative equations (Eqs (C2-4, C2-5)) must be used in the formation of the Vandermonde matrix. If the Vandermonde matrix was not defined this way, any system having repeated eigenvalue would have a singular, and thus

uninvertable, Vandermonde matrix. Csáki's generalized algorithm never forms the Vandermonde matrix, but incorporates the derivative feature directly in the formation of the inverse Vandermonde matrix.

Nearly Identical or Nearly Zero Eigenvalues. Eigenvalues that are close to each other or close to the origin may cause numerical instability because of the finite word length of digital computers. Algorithms using differences of eigenvalues, which Csáki's generalized algorithm does, are especially susceptible to this problem. The accuracy tests of this thesis therefore include some systems with close eigenvalues and some nearly zero eigenvalues.

Eigenvalues With Large Magnitude Differences. Combinations of eigenvalues with large differences in magnitude may cause scaling problems in the calculations. Some tests to investigate this problem are also included in the thesis.

Csáki's Generalized Algorithm

F. G. Csáki proposed (Ref 1:154-157) an algorithm for finding the inverse of a Vandermonde matrix directly from a collection of eigenvalues and their multiplicities. The Vandermonde matrix itself is never formed by the algorithm. Since the algorithm finds each element of the inverse Vandermonde matrix independently, it may be implemented on a parallel architecture computer to increase the real-time computation speed. This is discussed in more detail in the

next part of this section. Many inversion techniques cannot be implemented in parallel fashion because they do simple row or column operations rather than operations involving individual elements.

Csáki's generalized algorithm partitions the inverse Vandermonde matrix into m blocks, W_i , ($i = 1, 2, \dots, m$), with one block for each distinct eigenvalue. Each block is dimensioned k_i rows by n columns, where k_i is the eigenvalue multiplicity and n is the system order. The Vandermonde inverse may thus be represented by

$$V^{-1} = W = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ \vdots \\ W_m \end{bmatrix} \quad \text{nxn} \quad (23)$$

where

$$W_i = \begin{bmatrix} W_1^{T(i)} \\ W_2^{T(i)} \\ \vdots \\ W_{k_i}^{T(i)} \end{bmatrix} \quad k_i \times n \quad (24)$$

in which $W_j^{T(i)}$ ($j = 1, 2, \dots, k_i$) are the row vectors of the block W_i .

With this formulation and nomenclature, each element of the inverse Vandermonde matrix may be determined from

$$w_{jl}^{(i)} = \frac{1}{(k_i - j)!} \frac{d^{(k_i - j)}}{ds^{(k_i - j)}} \frac{N_l(s)}{D_i(s)} \bigg|_{s=\lambda_i} \quad (25)$$

in which $w_{jl}^{(i)}$ represents the element in the j 'th row and l 'th column of the i 'th block, W_i . The fraction terms are defined as follows:

$$D_i(s) = \frac{D(s)}{(s - \lambda_i)^{k_i}}; (i = 1, 2, \dots, m) \quad (26)$$

and

$$N_l(s) = s^{n-1} + a_{n-1}s^{n-l-1} + \dots + a_{l+1}s + a_l; \quad (27)$$

$$(l = 1, 2, \dots, n)$$

$D(s)$ is the system characteristic equation, which may be represented as

$$D(s) = s^n + a_{n-1}s^{n-1} + \dots + a_1s + a_0 = \prod_{i=1}^m (s - \lambda_i)^{k_i} \quad (28)$$

Eq (26), the i 'th "denominator polynomial," is therefore the characteristic equation with the i 'th eigenvalue divided out completely. The terms defined by Eq (27) are called "truncated polynomials" and are formed by iteratively reducing by one the power of each term of the characteristic equation, Eq (28), and deleting the right-

most term until a total of n truncated polynomials are found. For example, suppose a third order system has the characteristic polynomial

$$D(s) = s^3 + 4s^2 - s + 3 \quad (29)$$

The $n = 3$ truncated polynomials are

$$N_1(s) = s^2 + 4s - 1 \quad (30a)$$

$$N_2(s) = s + 4 \quad (30b)$$

$$N_3(s) = 1 \quad (30c)$$

Note that the final truncated polynomial is always the integer one because of the form of $D(s)$ used. Also notice from Eq (25) that the derivative procedure which is part of the definition of the Vandermonde matrix (see Appendix C, Section 2) is included in the formula for the elements of the inverse matrix.

In order to draw the foregoing together, suppose a sixth order system has three eigenvalues, having multiplicities of three, two, and one, respectively. Substituting Eq (25) into Eq (24), and the result into Eq (23) yields the following structure for the inverse Vandermonde matrix:

$$V^{-1} = W = \begin{bmatrix} W_1 \\ - \\ W_2 \\ - \\ W_3 \end{bmatrix} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} & W_{15}^{(1)} & W_{16}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} & W_{25}^{(1)} & W_{26}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} & W_{33}^{(1)} & W_{34}^{(1)} & W_{35}^{(1)} & W_{36}^{(1)} \\ - & - & - & - & - & - \\ W_{11}^{(2)} & W_{12}^{(2)} & W_{13}^{(2)} & W_{14}^{(2)} & W_{15}^{(2)} & W_{16}^{(2)} \\ W_{21}^{(2)} & W_{22}^{(2)} & W_{23}^{(2)} & W_{24}^{(2)} & W_{25}^{(2)} & W_{26}^{(2)} \\ - & - & - & - & - & - \\ W_{11}^{(3)} & W_{12}^{(3)} & W_{13}^{(3)} & W_{14}^{(3)} & W_{15}^{(3)} & W_{16}^{(3)} \end{bmatrix} \quad (31)$$

A simple numerical example of the application of this algorithm is in Appendix D, Example 3.

Parallel Architecture Processors and Csáki's Generalized Algorithm

If a large problem can be broken up into a number of smaller independent ones, parallel processing can be used to reduce the real process time. No computational advantage is realized, but since several calculations can be done simultaneously, the clock time from start to finish of the large problem is less than if each calculation were done serially. This is particularly important for flight control applications where real processing time may be critical.

For a comparison between serial and parallel process times, suppose a third order system has two eigenvalues, one with a multiplicity of two. A serial processor would perform the Vandermonde inversion with Csáki's algorithm

as shown in Figure 2a. If each calculation takes p seconds of processing time, the minimum total time is $15p$ because a calculation cannot begin until the preceding one is finished. Figure 2b shows the same calculations as a parallel processor could perform them. Notice that the total time from start to finish of the group of calculations is a third of the amount of time the serial processing took because the parallel processing system has three times as many processors to do the job. More information on parallel processing is found in Ref 4.

Not only does Csáki's method for finding the inverse of the Vandermonde matrix lend itself to parallel processing, so does the component matrix approach itself, as Figure 3 shows. At the top of the figure is $\hat{\theta}$, the current estimate of the system parameter vector. From that estimate, the system plant model, $\sigma(A)$, can be found, and from it, the current eigenvalues, $\lambda_1, \lambda_2, \dots, \lambda_n$, can be determined. As shown, each eigenvalue can be used in parallel to compute the $1 \times 2nr$ vector, F , of quadrature integrals, while at the same time computing the elements of the inverse Vandermonde matrix which correspond to that eigenvalue. At this point, the value of the parallel computation structure of Csáki's algorithm is readily visible. If the algorithm to find the inverse Vandermonde matrix did not have a parallel structure, none of the calculations using the elements of the inverse Vandermonde matrix could proceed until the entire matrix was computed in serial form.

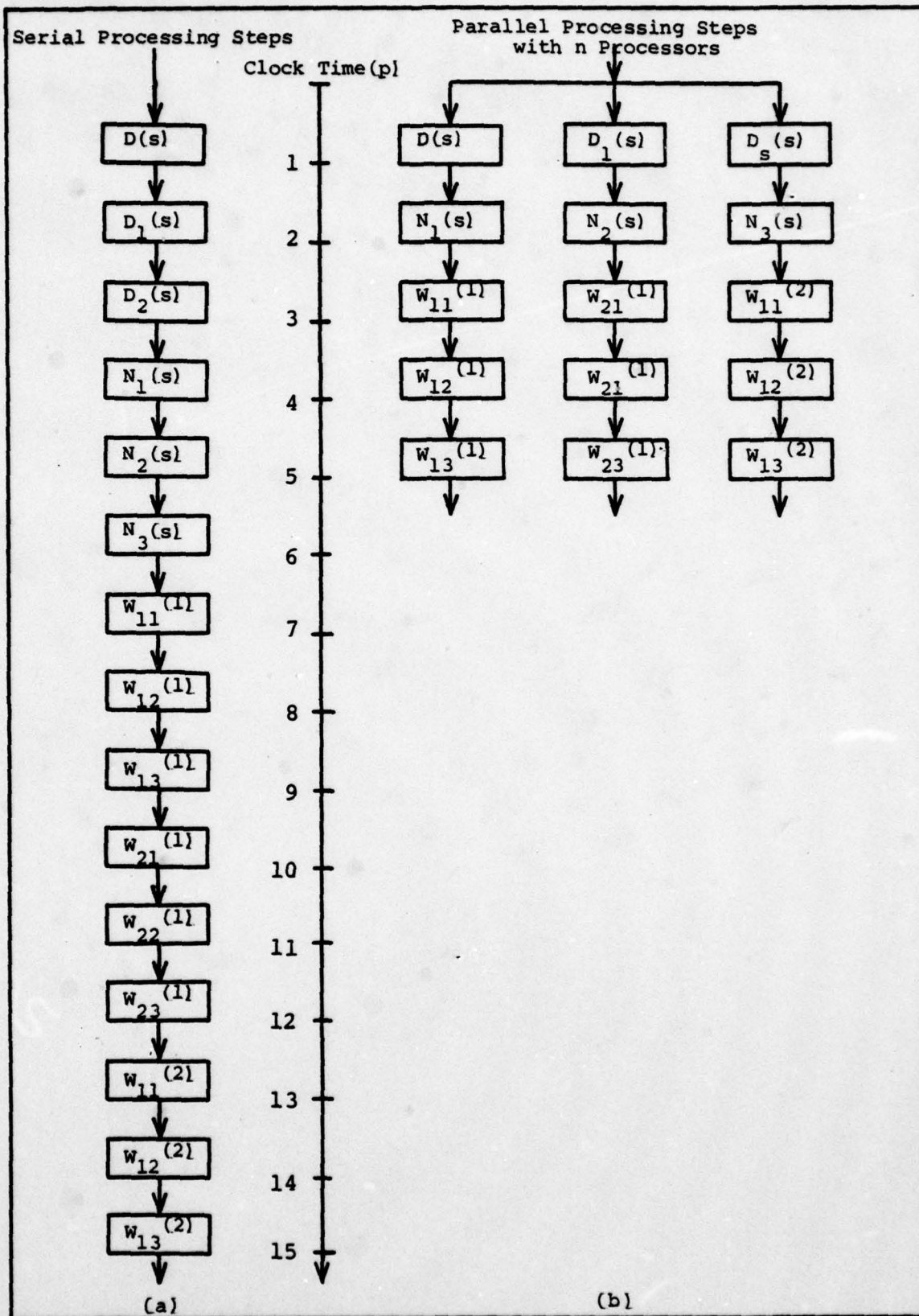


Figure 2. Processing time comparison:
a) Serial Processor,
b) Parallel Processor.

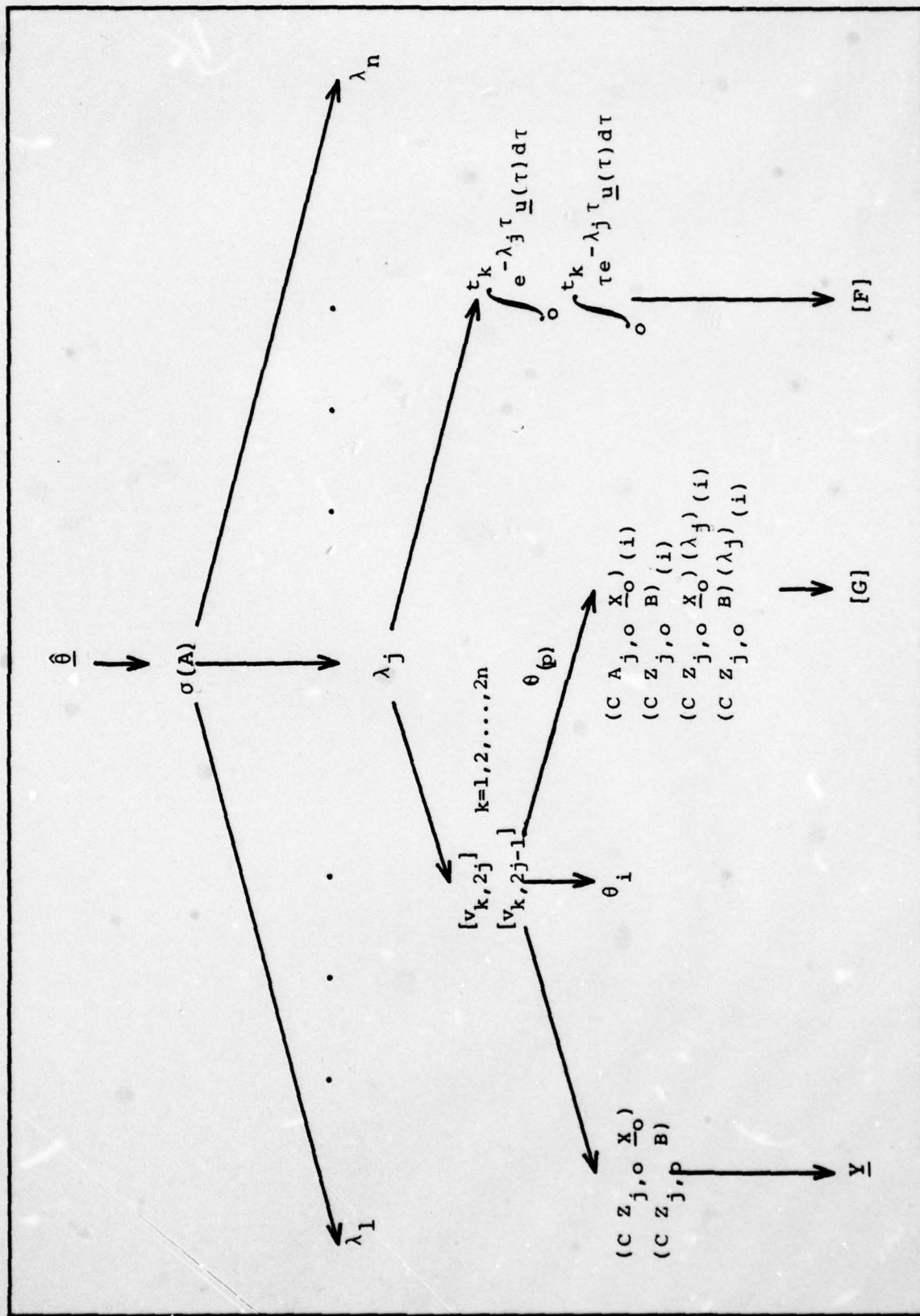


Figure 3. Parallel computation structure of the component matrix method of system identification. (Adapted from slides used to present Ref 10.)

That could stretch out the overall computation cycle time considerably, as Figure 2 showed. However, with the parallel structure of Csáki's algorithm, each eigenvalue branch of Figure 2 may find the elements of the inverse Vandermonde matrix pertaining to that eigenvalue and continue with the next calculations immediately. From the calculated inverse Vandermonde matrix elements, the vector of component matrices, \underline{Y} , and matrix of component sensitivities, G , can be computed. A change in the system parameters, $\Delta\theta$, can then be computed from (Ref 10)

$$\underline{Y} = [F] \cdot [G]^T \Delta\theta \quad (32)$$

where \underline{Y} , $[F]$, and $[G]$ are defined as in Figure 3.

It is now possible to use $\Delta\theta$ to update the estimate of the parameter vector, $\hat{\theta}$. This closes the computational loop, making the system identification process iterative, and thereby capable of applications in on-line flight controllers.

All of this sounds nice in theory. The immediate question that arises concerns whether or not the method actually works when implemented on a computer with a practical problem like limited word length which causes truncation and round off errors in the calculations performed. Therefore Csáki's algorithm was encoded in Fortran and some tests were run to evaluate its accuracy and efficiency. Those tests are the subject of the next section.

IV Tests Run on the Coded Version of

Csáki's Algorithm

Previous sections of this thesis dealt with the theory behind a system identification process. The CM method for system identifications was shown to have theoretical computational advantages over the more straightforward CLDE method. However, part of the CM method is the inversion of a $2n \times 2n$ Vandermonde matrix, which can be computationally costly. Csáki's generalized algorithm for finding the inverse Vandermonde matrix was chosen for its parallel computation structure. This structure gives the algorithm the capability of operating much faster in real time than algorithms which must perform all the calculations in serial form. Also, the particular parallel structure of Csáki's algorithm is very compatible with the parallel structure of the CM method of system identification.

Csáki's generalized algorithm was coded in Fortran in a subroutine called "VANINV", and is hereafter referred to as such. Appendix A contains descriptions and flowcharts of VANINV and the subroutines associated with it. Part I of Appendix B contains source listings of all the subroutines in Appendix A. This section describes tests done subroutines. Source listings of the test programs, though not flowcharted, are found in Appendix B, Part II.

Types of Tests

Two types of tests were performed on VANINV. The first was actually a combination of four separate tests to determine VANINV's computational accuracy. The second type of test measured how long it took VANINV to find the total inverse Vandermonde matrix for a given system. These two types of tests were run on a number of different systems to determine what happens to the computation time and accuracy when: (1) the system order increases, (2) two eigenvalues become very close, causing the Vandermonde matrix to become more nearly singular and therefore harder to invert, (3) one eigenvalue approaches zero, which also makes the Vandermonde matrix more nearly singular, and (4) the difference between the magnitudes of the largest and smallest eigenvalue increases. The ratio, $C = \left| \frac{\lambda_{\max}}{\lambda_{\min}} \right|$, in (4) is called the "condition index."

In order to obtain a relative measure of VANINV's performance, the same tests with the same systems were run using a standard matrix inversion routine contained in the International Mathematical & Statistical Library (IMSL). Early tests were done with IMSL subroutine LINV1F, but it did not have sufficient accuracy, so the final tests were done using IMSL subroutine LINV2F, which has a capability for iterative improvement of the solution.

Accuracy Tests

The first accuracy test is actually only a test of the IMSL routine. It simply involves feeding the inverse

Vandermonde matrix back into the routine, and comparing the reinverted matrix to the original Vandermonde matrix. It is unlikely that the accuracy of the reinverted matrix is better than that of the first inversion. Thus a lower accuracy bound on the inversion is the minimum number of decimal places in the reinverted matrix that are identical to the original matrix.

Originally a second accuracy test was set up to multiply the Vandermonde matrix by the calculated inverse and subtract the identity matrix:

$$V V_C^{-1} - I = E \quad (33)$$

in which the subscript "c" means "calculated." Theoretically, E should always be a null matrix. The extent to which it is not zero gives an indication of the accuracy of the calculation of the inverse, V_C^{-1} . However, E is typically composed of numbers so small that the errors caused by the computational characteristics of the algorithm cannot be distinguished from the truncation and round off errors resulting from the finite word length of the computer.

The following analysis may be made of the problem (Ref 6:61-64). What is really desired is the error between the true inverse and the actual inverse, defined as:

$$A_C^{-1} - A^{-1} = \text{Error} \quad (34)$$

Now

$$A(A_C^{-1} - A^{-1}) = AA_C^{-1} - AA^{-1} = AA_C^{-1} - I = E \quad (35)$$

from Eq (33). Then Eq (34) can be expressed

$$(A_C^{-1} - A^{-1}) = \text{Error} = A^{-1}(AA_C^{-1} - I) = A^{-1}E \quad (36)$$

Tests with matrices having known inverses show that the absolute value of the smallest negative exponent in the elements of $A^{-1}E$ approximates the minimum number of decimal places in A_C^{-1} that are accurate. For example, if the elements in $A^{-1}E$ are .xxxxxE-10, the elements of A_C^{-1} may be considered accurate to at least ten decimal places. This may be expressed as follows:

let α_{ij} be the elements of A^{-1}
 e_{ij} be the elements of E
 β_{ij} be the elements of $A^{-1}E$

$$\text{Therefore} \quad \beta_{ij} = \sum_{k=1}^n \alpha_{ik} e_{kj} \quad (37)$$

and a bound on the error of α_{ij} is given by

$$|\beta_{ij}| \leq \sum_{k=1}^n |\alpha_{ik}| |e_{kj}| \quad (38)$$

However, $\alpha_{ik} \in A^{-1}$ is not available; so use $\gamma_{ik} \in A_C^{-1}$ and use the following approximation for the error bound:

$$|\rho_{ij}| \leq \sum_{k=1}^n |\gamma_{ik}| |e_{kj}| \quad (39)$$

hoping that the actual error of the element is bounded by ρ_{ij} :

$$|\alpha_{ik} - \gamma_{ik}| < |\rho_{ij}| \quad (40)$$

When this test was applied to systems with known solutions, Eq (40) predicted inversion accuracy to within three decimal places for systems whose condition index was less than about 100, unless there were two eigenvalues closer than .2 units to each other.

The third test used was simply a summation of the elements, β_{ij} , of the check matrix, $A^{-1}E$, from test two. It was an attempt to quantify in a single number the accuracy of the inversion method being tested. In this thesis it is referred to as an "Accuracy Merit Figure" (AMF). The problem with it is that it is very dependent on how many elements of A_c^{-1} are nonzero. It tends, therefore, to be a cumulative, rather than an absolute (i.e.: "decimal place") accuracy test. Note, however, that the AMF can be used in conjunction with the largest element of the check matrix to carry the information contained in the matrix in two numbers rather than the n^2 elements of the matrix. Tests showed that if the AMF is much larger than the largest element of the check matrix (i.e.: $AMF \gg (\beta_{ij})_{\max}$), it indicates that many elements of the matrix are nearly the same size as $(\beta_{ij})_{\max}$ and probably all the elements of the inverse matrix, V^{-1} , are accurate to the same number of decimal places.

However, if $AMF \approx (\beta_{ij})_{\max}$, one element of the check matrix is dominant, and most of the other elements of V^{-1} are more accurate than $(\beta_{ij})_{\max}$ indicates. In this situation, $(\beta_{ij})_{\max}$ is definitely a worst case error indicator.

Fourth and final of the accuracy tests used was a subjective comparison of the inverse Vandermonde matrix calculated by VANINV and the IMSL routine. A number of the systems used in the tests had known solutions, or had repeating decimals in the solution. The decimal place accuracy of each inversion was therefore estimated on the basis of these characteristics of the solutions.

A total of 43 different systems were designed in order to test VANINV and the IMSL routines. The tests were run on a CDC 6600 computer using single precision arithmetic. Results of the tests are given in the next section.

V Results of Tests

The total of 43 test systems (see Table I) which were run were divided among the following five application categories:

1. Computation time as a function of system order (Tests 1 through 43).
2. Accuracy as a function of system order (Tests 1 through 11).
3. Accuracy as a function of the distance between the closest two eigenvalues of the system,
 $|\lambda_x - \lambda_y|_{\min}$ (Tests 12 through 20).
4. Accuracy as a function of the condition index,
 $C = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$ (Tests 21 through 27).
5. Accuracy as a function of increasing condition index, C , as the smallest eigenvalue, $|\lambda_{\min}|$, of the system approaches zero (Tests 28 through 43).

Table II contains all the data from the tests, and the results are explained according to the categories above.

Computation Time

Tests 1 through 11 indicate that the computation time for finding the inverse Vandermonde matrix is proportional to a number between n^2 and n^3 . This is to be expected since the number of elements to be computed is n^2 and some elements take more calculations than others. Figure 4 shows this relation (computation time between $n = 8$ and

Table I
Test Systems

| Test No. | System Order | Eigenvalues | | | |
|----------|--------------|---|--|--|--|
| 1 | 2 | $\lambda_1 = .5+j0.$ $\lambda_2 = -.5+j0.$ | | | |
| 2 | 3 | $\lambda_1 = .5+j0.$ $\lambda_2 = .5+j0.$ $\lambda_3 = 0.$ | | | |
| 3 | 4 | $\lambda_1 = 1. +j0.$ $\lambda_3 = .5+j0.$ $\lambda_2 = 1. +j0.$ $\lambda_4 = .5+j0.$ | | | |
| 4 | 6 | $\lambda_1 = 1.5+j0.$ $\lambda_3 = 1. +j0.$ $\lambda_5 = .5+j0.$ $\lambda_2 = -1.5+j0.$ $\lambda_4 = -1. +j0.$ $\lambda_6 = -.5+j0.$ | | | |
| 5 | 8 | $\lambda_1 = 2. +j0.$ $\lambda_3 = 1.5+j0.$ $\lambda_5 = 1. +j0.$ $\lambda_7 = .5+j0.$ $\lambda_2 = -2. +j0.$ $\lambda_4 = -1.5+j0.$ $\lambda_6 = -1. +j0.$ $\lambda_8 = -.5+j0.$ | | | |
| 6 | 10 | $\lambda_1 = 2.5+j0.$ $\lambda_4 = -2. +j0.$ $\lambda_6 = 1. +j0.$ $\lambda_9 = -.5+j0.$ $\lambda_2 = -2.5+j0.$ $\lambda_5 = 1.5+j0.$ $\lambda_7 = -1. +j0.$ $\lambda_3 = 2. +j0.$ $\lambda_6 = -1.5+j0.$ $\lambda_8 = .5+j0.$ | | | |
| 7 | 12 | $\lambda_1 = 3. +j0.$ $\lambda_4 = 2.5+j0.$ $\lambda_7 = 1.5+j0.$ $\lambda_{10} = -1. +j0.$ $\lambda_2 = -3. +j0.$ $\lambda_5 = 2. +j0.$ $\lambda_8 = -1.5+j0.$ $\lambda_{11} = .5+j0.$ $\lambda_3 = 2.5+j0.$ $\lambda_6 = -2. +j0.$ $\lambda_9 = 1. +j0.$ $\lambda_{12} = -.5+j0.$ | | | |
| 8 | 14 | $\lambda_1 = 3.5+j0.$ $\lambda_5 = 2.5+j0.$ $\lambda_9 = 1.5+j0.$ $\lambda_{13} = .5+j0.$ $\lambda_2 = -3.5+j0.$ $\lambda_6 = -2.5+j0.$ $\lambda_{10} = -1.5+j0.$ $\lambda_{14} = -.5+j0.$ $\lambda_3 = 3. +j0.$ $\lambda_7 = 2. +j0.$ $\lambda_{11} = 1. +j0.$ $\lambda_4 = -3. +j0.$ $\lambda_8 = -2. +j0.$ $\lambda_{12} = -1. +j0.$ | | | |
| 9 | 16 | $\lambda_1 = 4. +j0.$ $\lambda_5 = 3. +j0.$ $\lambda_9 = 2. +j0.$ $\lambda_{13} = 1. +j0.$ $\lambda_2 = -4. +j0.$ $\lambda_6 = -3. +j0.$ $\lambda_{10} = -2. +j0.$ $\lambda_{14} = -1. +j0.$ $\lambda_3 = 3.5+j0.$ $\lambda_7 = 2.5+j0.$ $\lambda_{11} = 1.5+j0.$ $\lambda_{15} = .5+j0.$ $\lambda_4 = -3.5+j0.$ $\lambda_8 = -2.5+j0.$ $\lambda_{12} = -1.5+j0.$ $\lambda_{16} = -.5+j0.$ | | | |
| 10 | 18 | $\lambda_1 = 4.5+j0.$ $\lambda_6 = -3.5+j0.$ $\lambda_{11} = 2. +j0.$ $\lambda_{16} = -1. +j0.$ $\lambda_2 = -4.5+j0.$ $\lambda_7 = 3. +j0.$ $\lambda_{12} = -2. +j0.$ $\lambda_{17} = .5+j0.$ $\lambda_3 = 4. +j0.$ $\lambda_8 = -3. +j0.$ $\lambda_{13} = 1.5+j0.$ $\lambda_{18} = -.5+j0.$ $\lambda_4 = -4. +j0.$ $\lambda_9 = 2.5+j0.$ $\lambda_{14} = -1.5+j0.$ $\lambda_5 = 3.5+j0.$ $\lambda_{10} = 2.5+j0.$ $\lambda_{15} = 1. +j0.$ | | | |
| 11 | 20 | $\lambda_1 = 5. +j0.$ $\lambda_6 = 4. +j0.$ $\lambda_{11} = 2.5+j0.$ $\lambda_{16} = -1.5+j0.$ $\lambda_2 = -5. +j0.$ $\lambda_7 = 3.5+j0.$ $\lambda_{12} = -2.5+j0.$ $\lambda_{17} = 1. +j0.$ $\lambda_3 = 4.5+j0.$ $\lambda_8 = -3.5+j0.$ $\lambda_{13} = 2. +j0.$ $\lambda_{18} = -1. +j0.$ $\lambda_4 = -4.5+j0.$ $\lambda_9 = 3. +j0.$ $\lambda_{14} = -2. +j0.$ $\lambda_{19} = .5+j0.$ $\lambda_5 = 4. +j0.$ $\lambda_{10} = -3. +j0.$ $\lambda_{15} = 1.5+j0.$ $\lambda_{20} = -.5+j0.$ | | | |

Table I--continued

| Test No. | System Order | Eigenvalues | |
|----------|--------------|--|--|
| 12 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4. + j0.$ |
| 13 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4.2 + j0.$ |
| 14 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4.4 + j0.$ |
| 15 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4.6 + j0.$ |
| 16 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4.8 + j0.$ |
| 17 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4.9 + j0.$ |
| 18 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4.99 + j0.$ |
| 19 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4.999 + j0.$ |
| 20 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4.9999 + j0.$ |
| 21 | 4 | $\lambda_1 = -100. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4. + j0.$ |
| 22 | 4 | $\lambda_1 = -1000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4. + j0.$ |
| 23 | 4 | $\lambda_1 = -10000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4. + j0.$ |
| 24 | 4 | $\lambda_1 = -100000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4. + j0.$ |
| 25 | 4 | $\lambda_1 = -1000000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4. + j0.$ |
| 26 | 4 | $\lambda_1 = -10000000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4. + j0.$ |
| 27 | 4 | $\lambda_1 = -100000000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -4. + j0.$ |
| 28 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -1. + j0.$ |
| 29 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_3 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.8 + j0.$ |

Table I--continued

| Test No. | System Order | Eigenvalues | |
|----------|--------------|---|---|
| 30 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.6 + j0.$ |
| 31 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.4 + j0.$ |
| 32 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.2 + j0.$ |
| 33 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.1 + j0.$ |
| 34 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.01 + j0.$ |
| 35 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.001 + j0.$ |
| 36 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.0001 + j0.$ |
| 37 | 4 | $\lambda_1 = -10. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = 0. + j0.$ |
| 38 | 4 | $\lambda_1 = -1000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -1. + j0.$ |
| 39 | 4 | $\lambda_1 = -1000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.1 + j0.$ |
| 40 | 4 | $\lambda_1 = -1000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.01 + j0.$ |
| 41 | 4 | $\lambda_1 = -1000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.001 + j0.$ |
| 42 | 4 | $\lambda_1 = -1000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = -.0001 + j0.$ |
| 43 | 4 | $\lambda_1 = -1000. + j0.$ $\lambda_2 = -9. + j0.$ | $\lambda_3 = -5. + j0.$ $\lambda_4 = 0. + j0.$ |

Table II
Test Results

| Test No. | System Order | No. of Iterations | Computation Time (Sec) | | Accuracy | | | | | | | | | | Con- dition No. | $ x-y $ minimum | $ x $ minimum |
|----------|--------------|-------------------|------------------------|-------|---------------------------|---------------------------|---|------|---|--|-----------------------------------|------------------------|--|------|-----------------------|-----------------|---------------|
| | | | | | Accuracy Merit Figure | | No. of Significant Digits in v^{-1} † | | No. of Significant Digits in $(v^{-1})^†$ | | Largest Non-Zero Element in CHECK | | Bound on No. of Significant Digits in v^{-1} † | | | | |
| | | | VARINV | INSL | VARINV | INSL | VARINV | INSL | (INSL ONLY) | | VARINV | INSL | VARINV | INSL | | | |
| 1 | 2 | 30 | .07 | .04 | 0. | 0. | 14 | 14 | 14 | | 0. | 0. | N/A | N/A | 1 | .5 | .5 |
| 2 | 3 | 30 | .14 | .06 | 0. | 0. | 14 | 14 | 14 | | 0. | 0. | N/A | N/A | 0 | .5 | .0 |
| 3 | 4 | 30 | .26 | .18 | 0. | 0. | 14 | 14 | 13 | | 0. | 0. | N/A | N/A | 2 | .5 | .5 |
| 4 | 6 | 30 | .60 | .33 | 8.21 $\times 10^{-14}$ | 1.10 $\times 10^{-13}$ | 14 | 14 | 13 | | 1.42 $\times 10^{-14}$ | 1.65 $\times 10^{-14}$ | 13 | 13 | 3 | .5 | .5 |
| 5 | 8 | 30 | 1.14 | .99 | 6.35 $\times 10^{-14}$ | 1.35 $\times 10^{-14}$ | 14 | 14 | 11 | | 1.03 $\times 10^{-14}$ | 1.03 $\times 10^{-14}$ | 13 | 13 | 4 | .5 | .5 |
| 6 | 10 | 30 | 1.88 | 1.48 | 2.54 $\times 10^{-12}$ | 2.54 $\times 10^{-12}$ | 14 | 14 | 12 | | 2.16 $\times 10^{-13}$ | 2.16 $\times 10^{-13}$ | 12 | 12 | 5 | .5 | .5 |
| 7 | 12 | 30 | 2.87 | 2.65 | 8.96 $\times 10^{-12}$ | 8.96 $\times 10^{-12}$ | 14 | 14 | 0* | | 6.48 $\times 10^{-13}$ | 6.48 $\times 10^{-13}$ | 12 | 12 | 6 | .5 | .5 |
| 8 | 14 | 30 | 4.13 | 3.87 | 2.86 $\times 10^{-11}$ | 2.87 $\times 10^{-11}$ | 14 | 14 | 10 | | 1.32 $\times 10^{-12}$ | 1.37 $\times 10^{-12}$ | 11 | 11 | 7 | .5 | .5 |
| 9 | 16 | 30 | 5.56 | 5.65 | 8.62 $\times 10^{-11}$ | 8.92 $\times 10^{-11}$ | 13 | 13 | 9 | | 3.74 $\times 10^{-12}$ | 4.33 $\times 10^{-12}$ | 11 | 11 | 8 | .5 | .5 |
| 10 | 18 | 30 | 7.43 | 7.45 | 9.72 $\times 10^{-6}$ | 2.35 $\times 10^{-10}$ | 13 | 13 | 8 | | 7.14 $\times 10^{-7}$ | 7.51 $\times 10^{-12}$ | 6 | 11 | 9 | .5 | .5 |
| 11 | 20 | 30 | 9.68 | 11.12 | 5.17 $\times 10^{-5}$ | 5.73 $\times 10^{-10}$ | 10 | 12 | 6 | | 3.96 $\times 10^{-6}$ | 2.02 $\times 10^{-11}$ | 5 | 10 | 10 | .5 | .5 |
| 12 | 4 | 100 | .86 | .82 | 1.68 $\times 10^{-12}$ | 1.68 $\times 10^{-12}$ | 14 | 14 | 12 | | 5.37 $\times 10^{-12}$ | 5.37 $\times 10^{-12}$ | 11 | 11 | 2.5 | 1. | 4.0 |
| 13 | 4 | 100 | .87 | .79 | 4.21 $\times 10^{-9}$ | 2.89 $\times 10^{-11}$ | 12 | 14 | 12 | | 7.43 $\times 10^{-10}$ | 9.96 $\times 10^{-13}$ | 9 | 11 | 2.4 | .8 | 4.2 |
| 14 | 4 | 100 | .86 | .77 | 7.92 $\times 10^{-10}$ | 2.69 $\times 10^{-11}$ | 14 | 13 | 11 | | 2.05 $\times 10^{-10}$ | 7.43 $\times 10^{-12}$ | 9 | 11 | 2.3 | .6 | 4.4 |
| 15 | 4 | 100 | .86 | .80 | 9.77 $\times 10^{-9}$ | 6.84 $\times 10^{-11}$ | 12 | 12 | 11 | | 2.75 $\times 10^{-9}$ | 2.15 $\times 10^{-11}$ | 8 | 10 | 2.2 | .4 | 4.6 |
| 16 | 4 | 100 | .86 | .80 | 7.00 $\times 10^{-8}$ | 5.02 $\times 10^{-10}$ | 14 | 13 | 11 | | 2.14 $\times 10^{-8}$ | 1.84 $\times 10^{-10}$ | 7 | 9 | 2.1 | .2 | 4.8 |
| 17 | 4 | 100 | .80 | .31 | 6.10 $\times 10^{-7}$ | 8.37 $\times 10^{-9}$ | 13 | 11 | 10 | | 1.99 $\times 10^{-7}$ | 1.58 $\times 10^{-9}$ | 6 | 8 | 2.0 | .1 | 4.9 |
| 18 | 4 | 100 | .87 | .32 | 1.55 $\times 10^{-1}$ | 5.72 $\times 10^{-7}$ | 10 | 10 | 9 | | 5.24 $\times 10^{-4}$ | 1.54 $\times 10^{-7}$ | 3 | 6 | 2.0 | .01 | 4.99 |
| 21 | 4 | 100 | .86 | .83 | 1.07 $\times 10^{-12}$ | 3.03 $\times 10^{-12}$ | 14 | 14 | 11 | | 2.82 $\times 10^{-13}$ | 1.32 $\times 10^{-12}$ | 12 | 11 | 25 | 1. | 4. |
| 22 | 4 | 100 | .87 | .77 | 1.21 $\times 10^{-12}$ | 1.21 $\times 10^{-12}$ | same | | 9 | | 5.67 $\times 10^{-13}$ | 5.67 $\times 10^{-13}$ | 12 | 12 | 2.3 $\times 10^4$ | 1. | 4. |
| 23 | 4 | 100 | .87 | .88 | 1.19 $\times 10^{-12}$ | 1.19 $\times 10^{-12}$ | same | | 7 | | 5.17 $\times 10^{-13}$ | 5.17 $\times 10^{-13}$ | 12 | 12 | 2.3 $\times 10^3$ | 1. | 4. |
| 24 | 4 | 100 | .86 | 1.12 | 1.82 $\times 10^{-12}$ | 1.82 $\times 10^{-12}$ | same | | 4 | | 6.96 $\times 10^{-13}$ | 6.96 $\times 10^{-13}$ | 12 | 12 | 2.3 $\times 10^4$ | 1. | 4. |
| 25 | 4 | 100 | .88 | 3.46 | 1.80 $\times 10^{-4}$ | 118. | 7 | 0 | 0* | | 9.00 $\times 10^{-3}$ | 88.4 | 4 | 0 | 2.3 $\times 10^5$ | 1. | 4. |
| 26 | 4 | 100 | .87 | 3.90 | .419 | 116. | 7 | 0 | 0* | | .210 | 88.4 | 0 | 0 | 2.3 $\times 10^6$ | 1. | 4. |
| 27 | 4 | 100 | .88 | .136 | 2.07 | 10 ²⁴ | 7 | 0 | 0* | | 9.00 $\times 10^{-7}$ | 10 ²⁴ | 6 | 0 | 2.3 $\times 10^7$ | 1. | 4. |

† By subjective observation.

* INSL had a terminal error.

‡ By second accuracy test.

Table II
Test Results

Table II--continued

| | | | | | | | | | | | | | | | | |
|----|---|-----|-----|-----|---------------------------|---------------------------|----|----|----|------------------------|------------------------|----|----|-----------------|----|-------|
| 28 | 4 | 100 | .89 | .92 | 7.49 $\times 10^{-13}$ | 2.62 $\times 10^{-12}$ | 14 | 14 | 10 | 1.85 $\times 10^{-13}$ | 9.81 $\times 10^{-13}$ | 12 | 12 | 10 | 1. | 1. |
| 29 | 4 | 100 | .87 | .77 | 2.52 $\times 10^{-13}$ | 2.08 $\times 10^{-12}$ | 12 | 14 | 11 | 5.66 $\times 10^{-11}$ | 7.26 $\times 10^{-13}$ | 10 | 12 | 12 | 1. | .8 |
| 30 | 4 | 100 | .86 | .78 | 2.91 $\times 10^{-10}$ | 1.01 $\times 10^{-12}$ | 12 | 14 | 11 | 5.62 $\times 10^{-11}$ | 2.94 $\times 10^{-13}$ | 10 | 12 | 16 | 1. | .6 |
| 31 | 4 | 100 | .86 | .78 | 1.97 $\times 10^{-10}$ | 1.72 $\times 10^{-12}$ | 12 | 14 | 10 | 3.99 $\times 10^{-11}$ | 6.35 $\times 10^{-13}$ | 10 | 12 | 25 | 1. | .4 |
| 32 | 4 | 100 | .88 | .79 | 1.36 $\times 10^{-10}$ | 4.76 $\times 10^{-13}$ | 13 | 14 | 9 | 2.99 $\times 10^{-11}$ | 9.11 $\times 10^{-14}$ | 10 | 13 | 50 | 1. | .2 |
| 33 | 4 | 100 | .88 | .79 | 1.79 $\times 10^{-10}$ | 5.02 $\times 10^{-13}$ | 10 | 14 | 8 | 3.76 $\times 10^{-11}$ | 1.38 $\times 10^{-13}$ | 10 | 12 | 10 ² | 1. | .1 |
| 34 | 4 | 100 | .84 | .79 | 9.39 $\times 10^{-11}$ | 7.34 $\times 10^{-13}$ | 9 | 12 | 6 | 2.97 $\times 10^{-11}$ | 2.06 $\times 10^{-13}$ | 10 | 12 | 10 ³ | 1. | .01 |
| 35 | 4 | 100 | .87 | .77 | 1.25 $\times 10^{-10}$ | 3.54 $\times 10^{-13}$ | 8 | 12 | 5 | 3.53 $\times 10^{-11}$ | 1.44 $\times 10^{-13}$ | 10 | 12 | 10 ⁴ | 1. | .001 |
| 36 | 4 | 100 | .86 | .80 | 6.35 $\times 10^{-11}$ | 6.37 $\times 10^{-13}$ | 5 | 10 | 3 | 1.91 $\times 10^{-11}$ | 1.74 $\times 10^{-13}$ | 10 | 12 | 10 ⁵ | 1. | .0001 |
| 37 | 4 | 100 | .85 | .30 | 2.04 $\times 10^{-13}$ | 1.54 $\times 10^{-12}$ | 13 | 13 | 12 | 6.20 $\times 10^{-14}$ | 5.31 $\times 10^{-13}$ | 13 | 12 | - | 1. | 0. |
| 38 | 4 | 100 | .86 | .79 | 3.65 $\times 10^{-14}$ | 3.67 $\times 10^{-14}$ | 14 | 14 | 9 | 1.16 $\times 10^{-14}$ | 1.16 $\times 10^{-14}$ | 13 | 13 | 10 ³ | 4. | 1. |
| 39 | 4 | 100 | .86 | .78 | 8.02 $\times 10^{-10}$ | 1.89 $\times 10^{-15}$ | 7 | 7 | 11 | 4.00 $\times 10^{-10}$ | 4.42 $\times 10^{-16}$ | 9 | 15 | 10 ⁴ | 4. | 1. |
| 40 | 4 | 100 | .88 | .80 | 1.42 $\times 10^{-9}$ | 9.79 $\times 10^{-15}$ | 7 | 7 | 9 | 7.08 $\times 10^{-10}$ | 4.47 $\times 10^{-15}$ | 9 | 14 | 10 ⁵ | 4. | .01 |
| 41 | 4 | 100 | .87 | .80 | 1.16 $\times 10^{-9}$ | 4.43 $\times 10^{-15}$ | 7 | 7 | 6 | 5.81 $\times 10^{-10}$ | 1.65 $\times 10^{-15}$ | 9 | 14 | 10 ⁶ | 4. | .001 |
| 42 | 4 | 100 | .87 | .80 | 1.30 $\times 10^{-9}$ | 4.12 $\times 10^{-15}$ | 7 | 7 | 4 | 6.47 $\times 10^{-10}$ | 1.63 $\times 10^{-15}$ | 9 | 14 | 10 ⁷ | 4. | .0001 |
| 43 | 4 | 100 | .87 | .72 | 1.13 $\times 10^{-15}$ | 1.15 $\times 10^{-15}$ | 14 | 14 | 10 | 1.02 $\times 10^{-15}$ | 1.02 $\times 10^{-15}$ | 14 | 14 | - | 4. | 0. |

+ By subjective observation.

* IMSL had a terminal error.

± By second accuracy test.

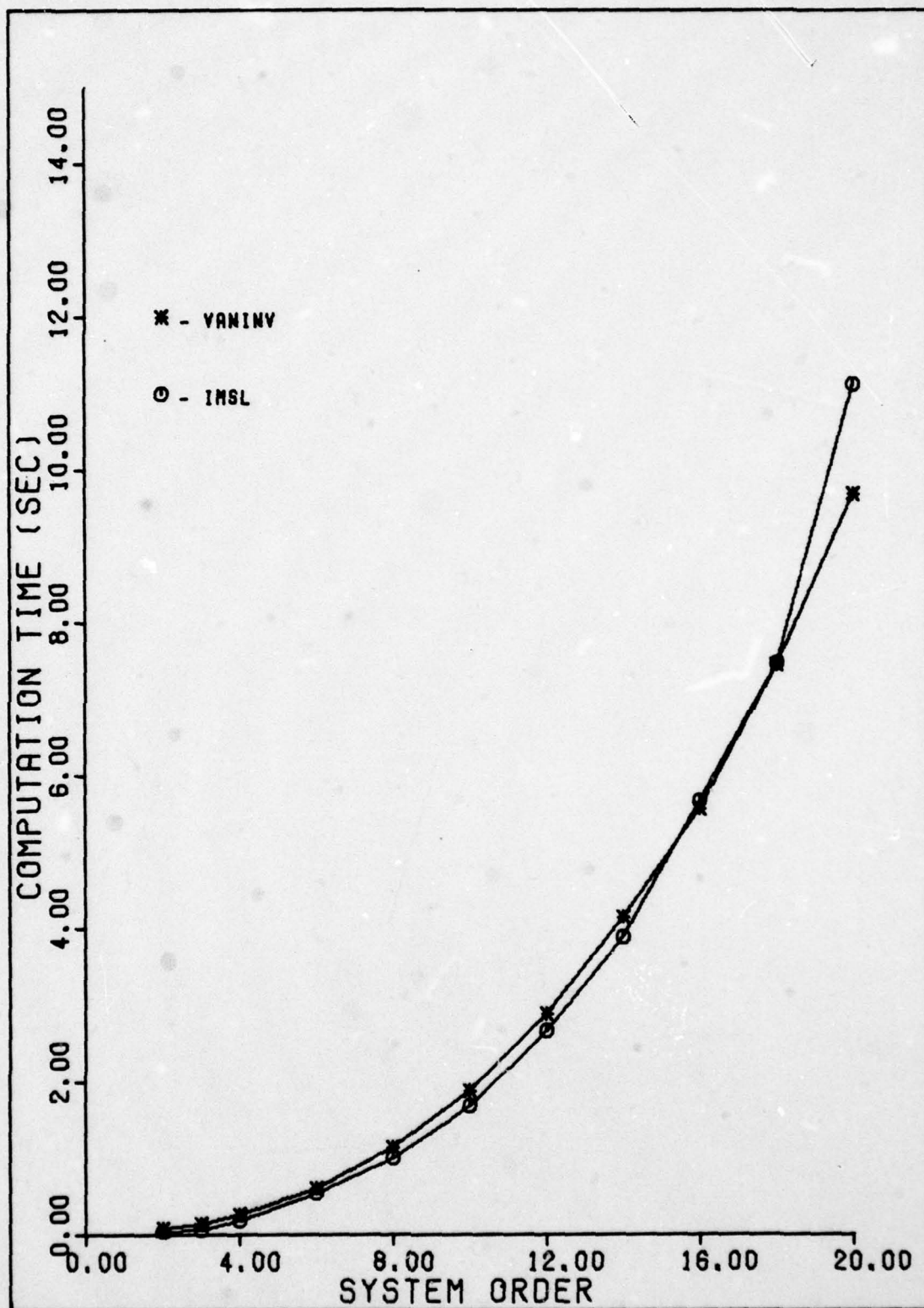


Figure 4. Computation time as a function. of system order (30 iterations with real eigenvalues).

$n = 16$ is proportional to about $n^{2.5}$) and also shows that the IMSL routines tended to run only slightly faster than VANINV. The remaining tests were all done with fourth order systems, and computation times remained nearly constant in every case for VANINV.

Accuracy as a Function of System Order

Figure 5 shows graphically what the results of Tests 1 through 11 indicate--accuracy is essentially constant at the maximum machine capability (14 decimal places for a CDC 6600 machine) over the range of system orders tested.

The second accuracy bound test indicates that although the full 14 decimal place accuracy is present up through the 14th order system, the probability of this being so is diminished as the system order increases. This may be explained as follows. The number of calculations done to find each element of the inverse Vandermonde matrix increases approximately to the n^2 to n^3 power with system order, as was shown in the preceding result. With the increase in the number of calculations, truncation and round off errors from the calculations may accumulate sufficiently to appear to propagate into the more significant decimal places. Since the second accuracy test uses absolute values of error quantities in determining a lower bound on the accuracy, the test is actually a calculation of the worst possible accuracy, not the most probable accuracy. This hypothesis would be supported if these tests were run in double precision arithmetic and the accuracy tracked along

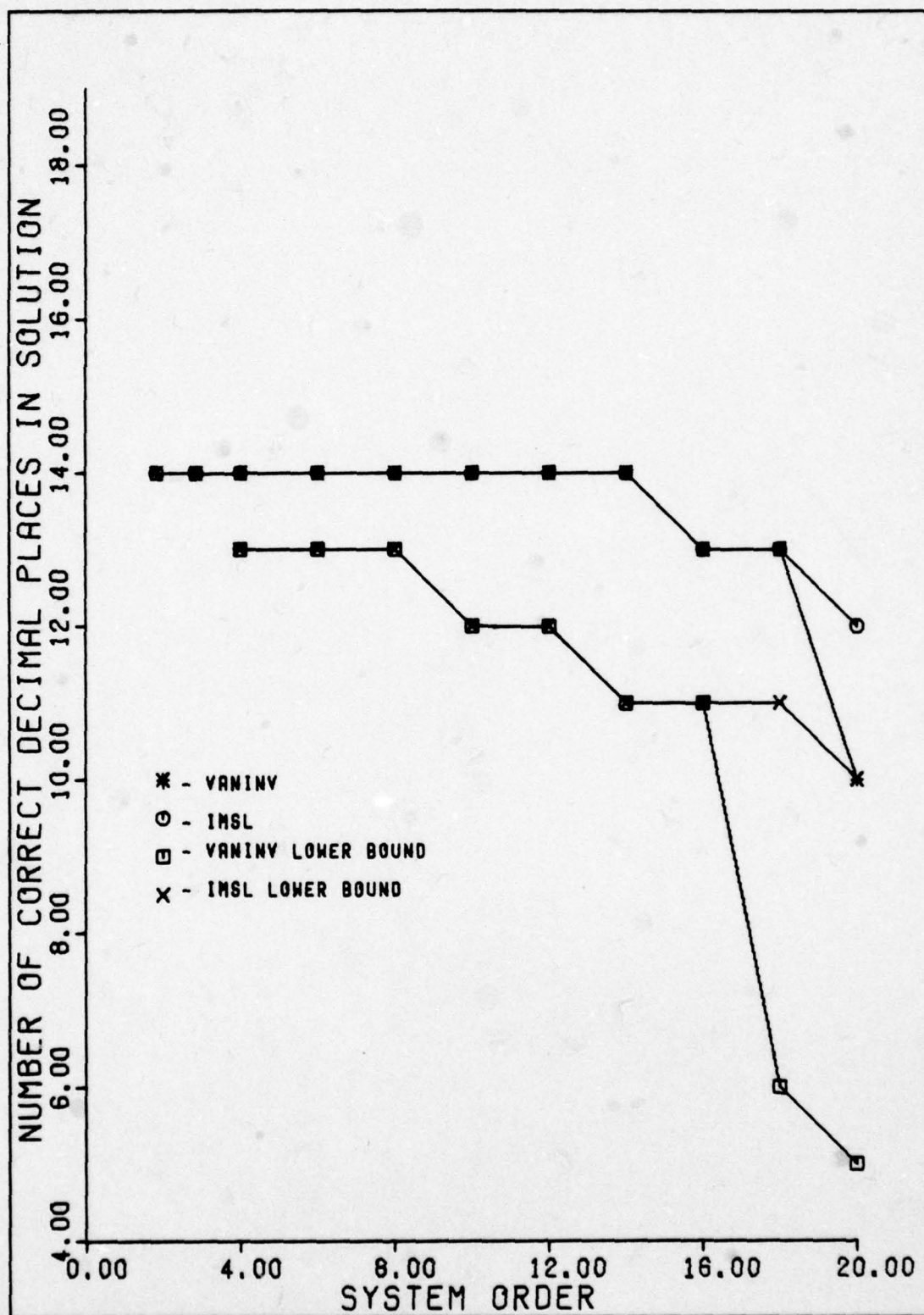


Figure 5. Computation accuracy as a function of system order (real eigenvalues).

at the higher level. The reason for the sudden drop in the minimum bound for VANINV beginning with the 18th order system is not intuitively obvious, but it is interesting to note that it seems to be indicative of a trend as the system order continues to increase.

Accuracy as a Function of $|\lambda_x - \lambda_y|_{\min}$

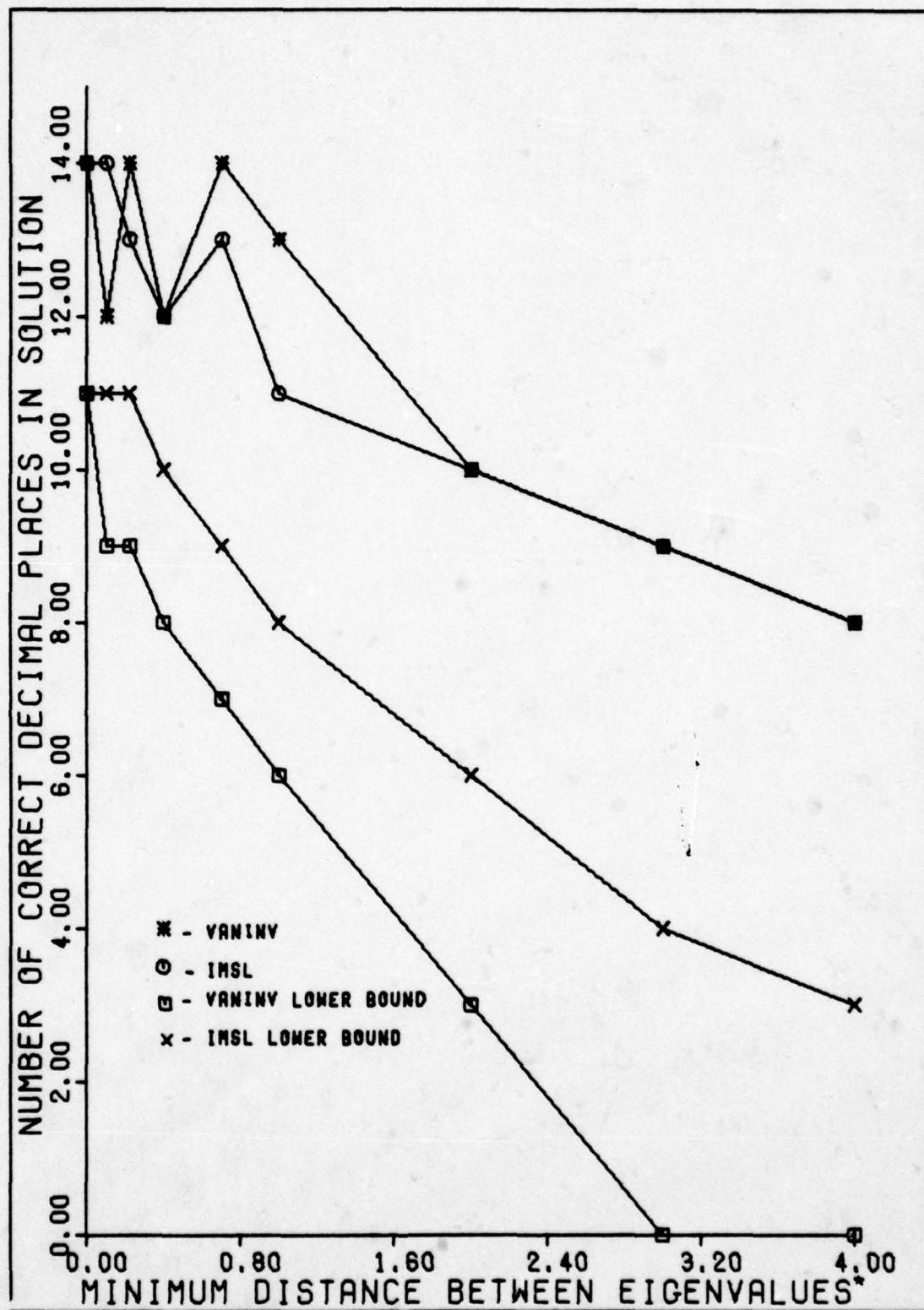
From Figure 6 it can be seen that the accuracy for both VANINV and the IMSL routine drops off as the minimum distance between two eigenvalues goes to zero. Neither routine is significantly better than the other, as the results of Tests 12 through 20 show.

Apparently the second accuracy bound test does not work very well for systems having two eigenvalues close to each other. The closer the eigenvalues become, the less realistic the accuracy bound is.

It is interesting to note that although the accuracy of the IMSL routine decreases as $|\lambda_x - \lambda_y|_{\min}$ decreases, the resulting inverse is stable. In each test the inverse and the inverse of the inverse have about the same accuracy, so the solution of the inverse matrix is termed "stable" for this discussion.

Accuracy as a Function of Condition Index

Tests 21 through 27 hold the minimum eigenvalue, $|\lambda_{\min}|$, and the closest distance, $|\lambda_x - \lambda_y|_{\min}$, between two eigenvalues constant as the condition index, $C = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$,



* $(-\log|\lambda_x - \lambda_y|_{\min})$

Figure 6. Computation accuracy as a function of the two closest eigenvalues.

increased. The reinversion of the inverse by the IMSL routine indicates that the inversion accuracy decreases as the condition index increases. The actual inversion accuracy could not be determined by inspection of the data, though VANINV and the IMSL routine gave identical results up through a condition index of twenty-five thousand. Beyond that point both inversion routines became unstable, as indicated by the large fluctuations in the elements of the check matrix and the Accuracy Merit Figure. Plots were not made of this test series because of the small amount of plotable data.

Accuracy as a Function of Increasing Condition Index
Caused by an Eigenvalue Approaching Zero

These tests were run in two series. The first series, Tests 28 through 37, allowed one eigenvalue to go to zero, thereby causing the eigenvalue condition index, C , to increase. The minimum distance between any two eigenvalues was held constant. As seen in Figure 7, the accuracy of both inversion routines dropped as the condition number increased.

In the second series of tests, numbers 38 through 43, the minimum distance between any two eigenvalues was again held constant, but at a larger value in order to determine if it has any effect on the inversion accuracy. The accuracy could not be determined by inspecting the inverse matrices found. However, based on the accuracy of the reinverted

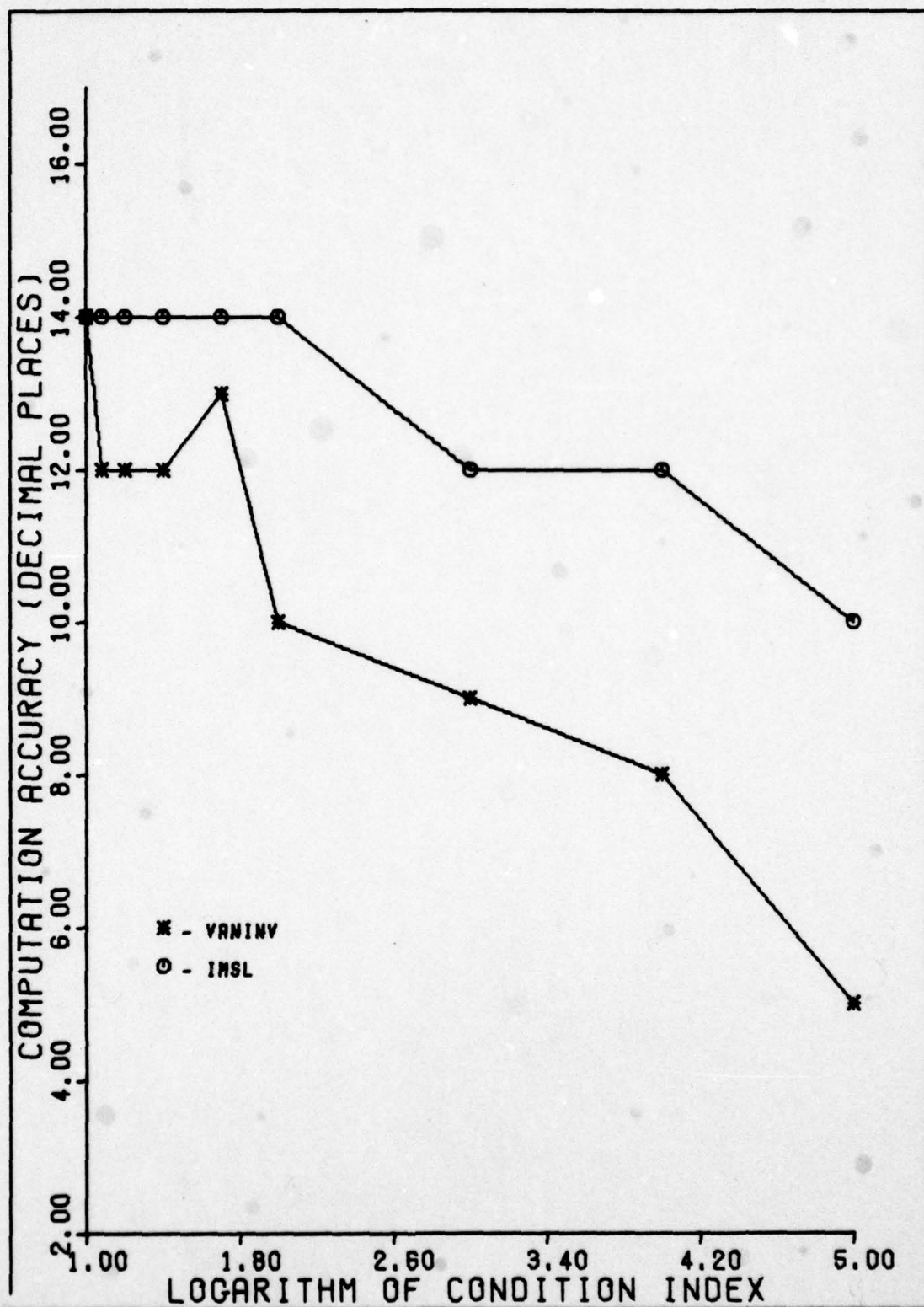


Figure 7. Logarithm of condition index ($\log |\lambda_{\max}| / |\lambda_{\min}|$).

matrix, the accuracy and stability of the IMSL routines dropped as the condition index grew in each series.

From the results of both series of tests, a high condition index resulting from an eigenvalue near zero is not as detrimental to inversion accuracy or stability as a high condition index caused by a large eigenvalue. In addition, the combination of close eigenvalues and a high condition index has more of a degrading effect on accuracy than either of the two by themselves. It may also be observed that when the eigenvalue actually reaches zero, thereby driving the condition index to infinity, the accuracy rises sharply back to the level realized for small condition indices. This is probably because the computer can represent zero exactly, whereas numbers near zero can only be approximated.

All of the findings of the tests are summarized in the several conclusions of the next section. In addition, recommendations are given for possible improvements to VANINV and for its use in further research regarding system identification processes.

VI Conclusions and Recommendations

Several conclusions were drawn from the results reported in Section V. Based on those conclusions it is recommended that subroutine VANINV, which implements Csáki's generalized algorithm for finding the inverse of a Vandermonde matrix, be refined for computational speed and tested in its intended application, that of finding component matrices for a system identification process. The conclusions and the recommendations are explained further in the remainder of this section. The method of presentation is a listing of the conclusions/recommendations, followed by some comments regarding them.

Conclusions

The following conclusions were made:

1. The IMSL routines are generally slightly faster, and about as accurate or somewhat more accurate than VANINV is in its present form.
2. The accuracy of both routines is reduced by high condition indices, particularly if the high condition indices are caused by large eigenvalues, as opposed to being caused by one eigenvalue near zero.

In regard to computation speed, it should be noted that VANINV is compiled to optimization level two, whereas the IMSL routines are only compiled to optimization level

one. This means that IMSL could possibly run significantly faster than VANINV. However, these tests were restricted to real distinct eigensystems. VANINV has the capability of computing the inverse of Vandermonde matrices with combinations of repeated and/or complex eigenvalues. A fair test of this property would necessitate special conversions to turn the Vandermonde matrix into a purely real matrix that the IMSL routines could handle. This conversion time would then be considered part of the IMSL computation time.

It would be interesting to determine VANINV's accuracy if double precision arithmetic was used. Of course, for any intended application a trade-off analysis should be performed between any increase in accuracy and the cost in terms of additional memory requirements.

Recommendations

It is recommended that VANINV be developed further and implemented in a component matrix approach for system identification. In this way, VANINV's accuracy can be checked in the more meaningful environment of its intended application. An accuracy check is built right into the component matrix approach to system identification: all of the component matrices of the pseudo modes should be null. Also, the computational accuracy should be tested on computers with shorter word length than possessed by the CDC 6600 computer used for these tests. Computers used for on-line control applications probably wouldn't have word lengths with 48 bits of data storage like the CDC 6600 has.

The following procedures are therefore recommended for VANINV itself:

1. Test VANINV with repeated and/or complex eigenvalues to determine their effect on computation time and accuracy.
2. Determine where most of the computation time is used on VANINV and improve the efficiency where practical.
3. Eliminate the calls to subroutines that are made only one or two places in VANINV and replace them with the coded subroutines, since all the subroutines have been debugged.
4. Streamline VANINV by using vector array addressing to reduce array index computation time.
5. Incorporate a row jumping capability into VANINV to eliminate duplicated calculations for repeated and/or complex eigenvalues. This is illustrated in Figure 8, which shows a very common occurrence for the component matrix approach. Every complex conjugate pair of eigenvalues in the original system will be repeated in the $2n \times 2n$ Vandermonde matrix of the augmented system. The resulting calculations that VANINV would perform for these eigenvalues is shown by Figure 8. Note that the lower half of the matrix shown in Figure 8 is simply the complex conjugate of the upper half. Thus considerable computation time could be saved

| | | | |
|---|---|---|---|
| $\left. \frac{d}{ds} \frac{N_1(s)}{D_1(s)} \right _{\lambda_1}$ | $\left. \frac{d}{ds} \frac{N_2(s)}{D_1(s)} \right _{\lambda_1}$ | $\left. \frac{d}{ds} \frac{N_3(s)}{D_1(s)} \right _{\lambda_1}$ | $\left. \frac{d}{ds} \frac{N_4(s)}{D_1(s)} \right _{\lambda_1}$ |
| $\left. \frac{N_1(s)}{D_1(s)} \right _{\lambda_1}$ | $\left. \frac{N_2(s)}{D_1(s)} \right _{\lambda_1}$ | $\left. \frac{N_3(s)}{D_1(s)} \right _{\lambda_1}$ | $\left. \frac{N_4(s)}{D_1(s)} \right _{\lambda_1}$ |
| ----- | | | |
| $\left. \frac{d}{ds} \frac{N_1(s)}{D_2(s)} \right _{\lambda_2}$ | $\left. \frac{d}{ds} \frac{N_2(s)}{D_2(s)} \right _{\lambda_2}$ | $\left. \frac{d}{ds} \frac{N_3(s)}{D_2(s)} \right _{\lambda_2}$ | $\left. \frac{d}{ds} \frac{N_4(s)}{D_2(s)} \right _{\lambda_2}$ |
| $\left. \frac{N_1(s)}{D_2(s)} \right _{\lambda_2}$ | $\left. \frac{N_2(s)}{D_2(s)} \right _{\lambda_2}$ | $\left. \frac{N_3(s)}{D_2(s)} \right _{\lambda_2}$ | $\left. \frac{N_4(s)}{D_2(s)} \right _{\lambda_2}$ |

Figure 8. Example of VANINV calculations applied to a fourth order system with a repeated complex conjugate pair of eigenvalues.

by incorporating a row jumper which would simply store the complex conjugate of

$$w_{1\ell}^{(1)} = \left. \frac{d}{ds} \frac{N_{\ell}(s)}{D_1(s)} \right|_{\lambda_1}$$

directly into

$$w_{1\ell}^{(2)} = \left. \frac{d}{ds} \frac{N_{\ell}(s)}{D_2(s)} \right|_{\lambda_2}$$

and thus eliminate half the calculations. Also note that a repeated eigenvalue in the original system would occur four times in the $2n \times 2n$ Vandermonde matrix, as Figure 9 shows. Each column is composed of identical elements except for the number of derivatives that must be taken for evaluation. If a row jumper with some memory were added to the routine, successive derivatives could be taken to evaluate the elements of each column. This would require only $1+1+1 = 3$ derivatives for each column shown rather than $3+2+1 = 6$ derivatives as VANINV must do presently. (Without the row jumper, VANINV simply evaluates each element as it comes to it, without regard for any calculations done previously.)

In addition to the recommended procedures for VANINV described above, two comments are made regarding the component matrix approach to system identification:

$$\begin{bmatrix}
 \left. \frac{d^3}{ds^3} \frac{N_1(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d^3}{ds^3} \frac{N_2(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d^3}{ds^3} \frac{N_3(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d^3}{ds^3} \frac{N_4(s)}{D_1(s)} \right|_{\lambda_1} \\
 \left. \frac{d^2}{ds^2} \frac{N_1(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d^2}{ds^2} \frac{N_2(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d^2}{ds^2} \frac{N_3(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d^2}{ds^2} \frac{N_4(s)}{D_1(s)} \right|_{\lambda_1} \\
 \left. \frac{d}{ds} \frac{N_1(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d}{ds} \frac{N_2(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d}{ds} \frac{N_3(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{d}{ds} \frac{N_4(s)}{D_1(s)} \right|_{\lambda_1} \\
 \left. \frac{N_1(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{N_2(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{N_3(s)}{D_1(s)} \right|_{\lambda_1} & \left. \frac{N_4(s)}{D_1(s)} \right|_{\lambda_1}
 \end{bmatrix}$$

Figure 9. Example of VANINV calculations applied to a fourth order system with like eigenvalues.

1. Many of the matrices resulting from the component matrix approach are very sparse (eg: Eqs (D2-13), (D2-14)), and thus the method lends itself readily to sparse matrix operations such as those available with SOFE (Ref 7).
2. The calculation structure of both the component matrix approach and VANINV lend themselves to parallel calculations. An implementation of these methods on a parallel architecture computer has the potential for being very fast in real time.

Thus it may be seen that a refinement of VANINV may well have a place in future system identification processes.

Bibliography

1. Csáki, F. G. "Some Notes on the Inversion of Confluent Vandermonde Matrices," IEEE Transactions on Automatic Control, AC-20 (1):154-157 (February 1975).
2. "Flight, History of," Encyclopaedia Britannica Macropaedia, 7:388 (1974).
3. Gaardabassi, G., K. Locatelli, and S. Rinaldi. "Structural Properties of Sensitivity Systems," Journal of the Franklin Institute, 294 (4):241-248 (October 1972).
4. Heller, Don. "A Survey of Parallel Algorithms in Numerical Linear Algebra," SIAM Review, 20 (4): 740-777 (October 1978).
5. Moler, Cleve and Charles Van Loan. "Nineteen Dubious Ways to Compute the Exponential of a Matrix," SIAM Review, 20 (4):801-836 (October 1978).
6. Moursund, David G. and Charles S. Duris. Elementary Theory and Application of Numerical Analysis. New York: McGraw-Hill Book Company, 1967.
7. Musick, Stanton H. SOFE: A Generalized Digital Simulation for Optimal Filter Evaluation User's Manual. Technical Memorandum AFAL-TM-78-19. Wright-Patterson AFB, Ohio: Air Force Avionics Lab, June 1978.
8. Reid, J. G., et al. "An Algebraic Representation of Parameter Sensitivity in Linear Time-invariant Systems," Journal of the Franklin Institute, 301 (1,2):123-141 (January-February 1976).
9. Reid, J. G. Lecture materials distributed in EE 5.10, Linear Methods of System Analysis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1977.
10. ----- "A New Parallel Identification Algorithm for Linear Time-invariant Systems: Preliminary Results," paper presented at the 2nd Annual Workshop Between Applied Mathematics & Industry, Naval Post Graduate School, Monterey, California, 20-24 February 1979.
11. ----- Sensitivity Operators and Associated System Concepts for Linear Dynamic Systems. PhD dissertation. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, July 1976.

12. Swisher, George M. Introduction to Linear System Analysis. Champaign, Illinois: Matrix Publishers, Inc., 1976.
13. Tomović, R. and M. Vukobratović. General Sensitivity Theory. New York: American Elsevier Publishing Company, Inc., 1972.
14. "Wright, Wilbur," Encyclopedia Americana, 29:557 (1968).
15. Zadeh, Lofti A. and Charles A. Desoer. Linear System Theory. New York: McGraw-Hill Book Company, 1963.

VITA

Donald Paul Seyler was born on 4 October 1952 in Niagara Falls, New York. He graduated from high school in Wilson, New York in 1971, and attended Houghton College until called to active duty in February 1973. Upon completing technical training, he served as a radio repairman in the 308th Communication Squadron, Little Rock AFB, Arkansas. In September of 1974 he was selected for the Airman Education and Commissioning Program. He received the degree of Bachelor of Science in Electrical Engineering in June 1977 from Ohio University. Upon graduation he received a commission in the USAF and a direct assignment to the School of Engineering, Air Force Institute of Technology.

Permanent address: 2863 Youngstown-Lockport Rd.
Ransomville, N. Y. 14131

APPENDIX A

COMPUTER PROGRAMS FOR CSÁKI'S
GENERALIZED INVERSE VANDERMONDE
ALGORITHM

Preface for Appendix A

This appendix contains the subroutines written to implement the generalized method for finding an inverse Vandermonde matrix that is described in the body of the report. The appendix is formatted to serve as a combination user/programmer guide for further investigation of the proposed alternate method of system identification which is outlined in the main text.

Contents and Subroutine Cross-reference

| Subroutine | Subroutines Called | | | | | | | | | | | | | | Description Page | Flowchart Page |
|--------------------|--------------------|---------|-------|------|---------|---------|-------|-------|---------|---------|--------|---------|-----|--------|---------------------|--|
| | CGAIN | COPYPOL | ESORT | EVAL | EXPANDC | FRACDIF | NFACT | OUT2C | POLYADD | POLYDIF | POLYMC | POLYSUB | RAT | ROOTAY | UNITY | VANINV |
| CGAIN | | | | | | | | | | | | | | | | A-4 A-5 |
| COPYPOL | | | | | | | | | | | | | | | | A-6 A-7 |
| ESORT [†] | | | | | | | | | | | | | | | | A-8 A-9 A-10 A-11 |
| EVAL | X | | | | | | | | | | | | | | | A-12 A-13 |
| EXPANDC | | | | | | | | | | | X | | | | | A-14 A-15 |
| FRACDIF | | | | | | | | | | X | X | X | | | | A-16 A-17 A-18 |
| NFACT [†] | | | | | | | | | | | | | | | | A-19 A-19 |
| OUT2C [†] | | | | | | | | | | | | | | | | A-20 A-21 |
| POLYADD | X | | | | | | | | | | | | | | X | A-22 A-23 A-24 |
| POLYDIF | X | | | | | | | | | | | | | | | A-25 A-26 |
| POLYMC | | | | | | | | | | | | | | | X | A-27 A-28 |
| POLYSUB | | | | | | | | | X | | | | | | | A-29 A-30 |
| RAT | | | | | | | | | | | | | | | | A-31 A-32 |
| ROOTAY | | | | | | | | | | | | | | | | A-33 A-34 |
| UNITY | | | | | | | | | | | | | | | | A-35 A-36 |
| VANINV | | X | | X | X | X | X | | | | | | X | X | | A-37 thru A-39 A-40 thru A-46 |

[†] Not used by any of the subroutines listed, but pertinent to the use of VANINV.

[†] Function subprogram.

Subroutine CGAIN

Subroutine CGAIN multiplies each coefficient of a polynomial by a specified gain. The coefficients and the gain may be complex.

Subroutine statement: SUBROUTINE CGAIN (AR,AI,NSA,AKR,AKI)

Subroutines called: None

Variables:

AR| = Arrays containing the real and imaginary parts of
AI| = the polynomial coefficients

NSA = Number of storage locations required for the
polynomial coefficients

AKR| = Real and imaginary parts of the polynomial gain
AKI|

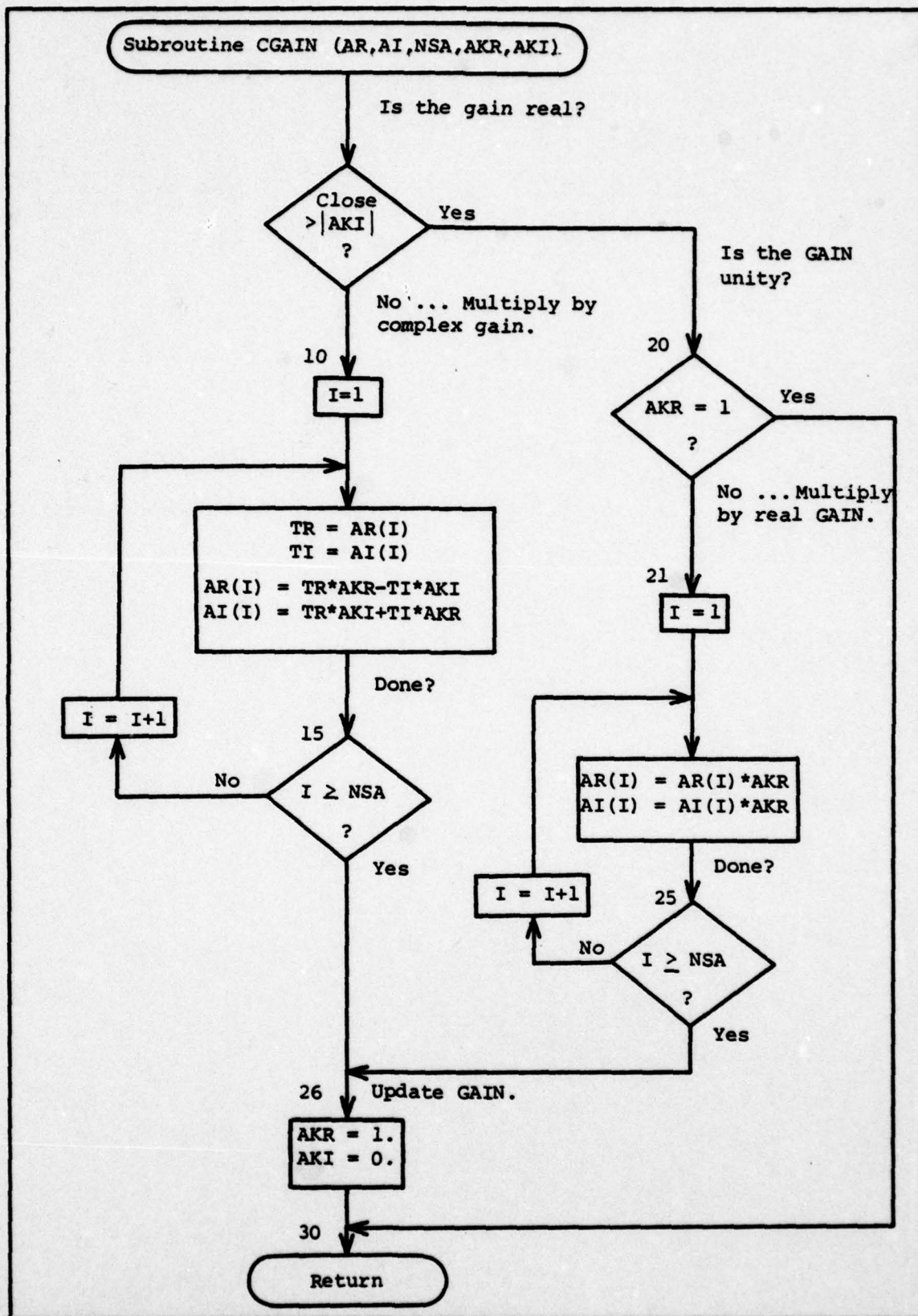


Figure A-1. Flowchart for subroutine CGAIN.

Subroutine COPYPOL

Subroutine COPYPOL copies the coefficients of the input polynomial into the storage locations for the coefficients of the output polynomial. Also, the input gain is copied into the output gain storage. The coefficients and gains may be complex.

Subroutine statement: SUBROUTINE COPYPOL (PIR,PII,GIR,GII,N,POR,POI,
GOR,GOI]

Subroutines called: None

Variables:

PIR)
PII) Arrays containing the real and imaginary parts of the
POR) coefficients of the input (PIx) and output (POx) polynomials
POI)

N = Number of storage locations needed for the polynomial
 coefficients. (Note that N is one integer value larger
 than the polynomial order.)

GIR)
GII) Real and imaginary parts of the gains of the input (GIx)
GOR) and output (GOx) polynomials
GOI)

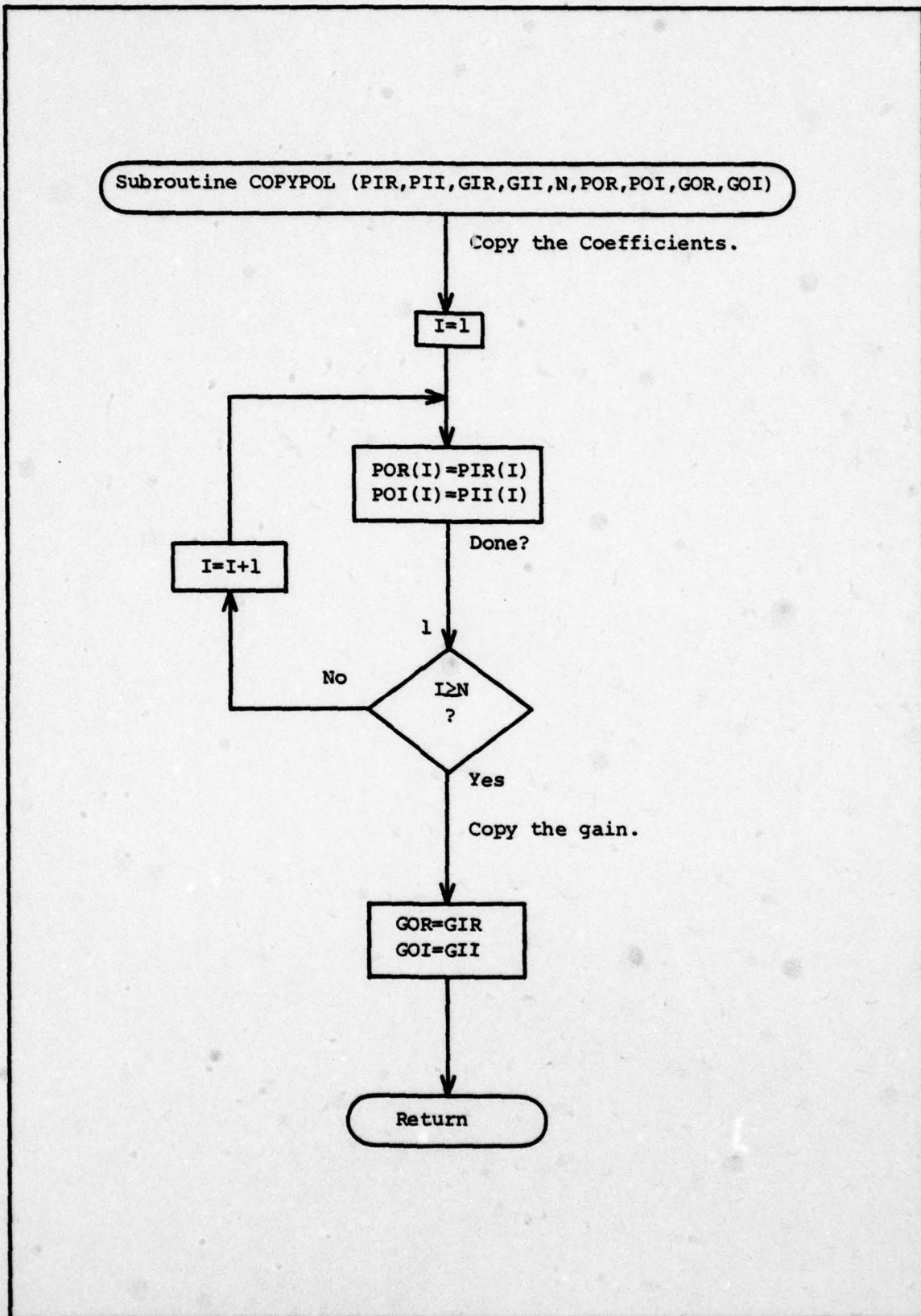


Figure A-2. Flowchart for subroutine COPYPOL.

Subroutine ESORT

Subroutine ESORT sorts a collection of real and complex eigenvalues according to the following specifications:

1. Put in order of decreasing complexness; i.e., complex before real.
2. Order each section in decreasing order of multiplicity.
3. Order complex conjugates with the eigenvalue having the positive imaginary part.
4. For equal multiplicities, order in terms of decreasing magnitude of the real part; or if equal, place eigenvalues with positive real parts before those with negative real parts, and order in terms of the imaginary part.

Subroutine ESORT sweeps through the collection and compares each pair of adjacent eigenvalues to determine whether or not they need to be swapped to get them in the proper relative order. Each time a swap is performed, a "swap counter" is incremented. At the end of each sweep through the eigenvalue array, the swap counter is tested to see if any swaps were made on that sweep. If there were swaps, the counter is reset to zero and another sweep through the array is started. Zero swaps during any sweep through the array indicates that all the eigenvalues are in the proper order, so the subroutine returns the rearranged eigenvalue array.

Subroutine statement: SUBROUTINE ESORT (EIGR,EIGI,KI,M)

Subroutines called: None

Variables:

EIGR) = Arrays containing the real and imaginary parts of
EIGI) the collection of eigenvalues

KI = Array containing the corresponding eigenvalue
multiplicities

M = Number of different eigenvalues in the array (complex conjugates are considered to be different)

MDIS = "Message DISable" flag, passed in blank common. ESORT normally prints an informative diagnostic if two identical eigenvalues are encountered. If the user doesn't want the message printed, setting MDIS equal to any positive integer will disable the printer. Setting MDIS to zero or a negative integer enables the printer.

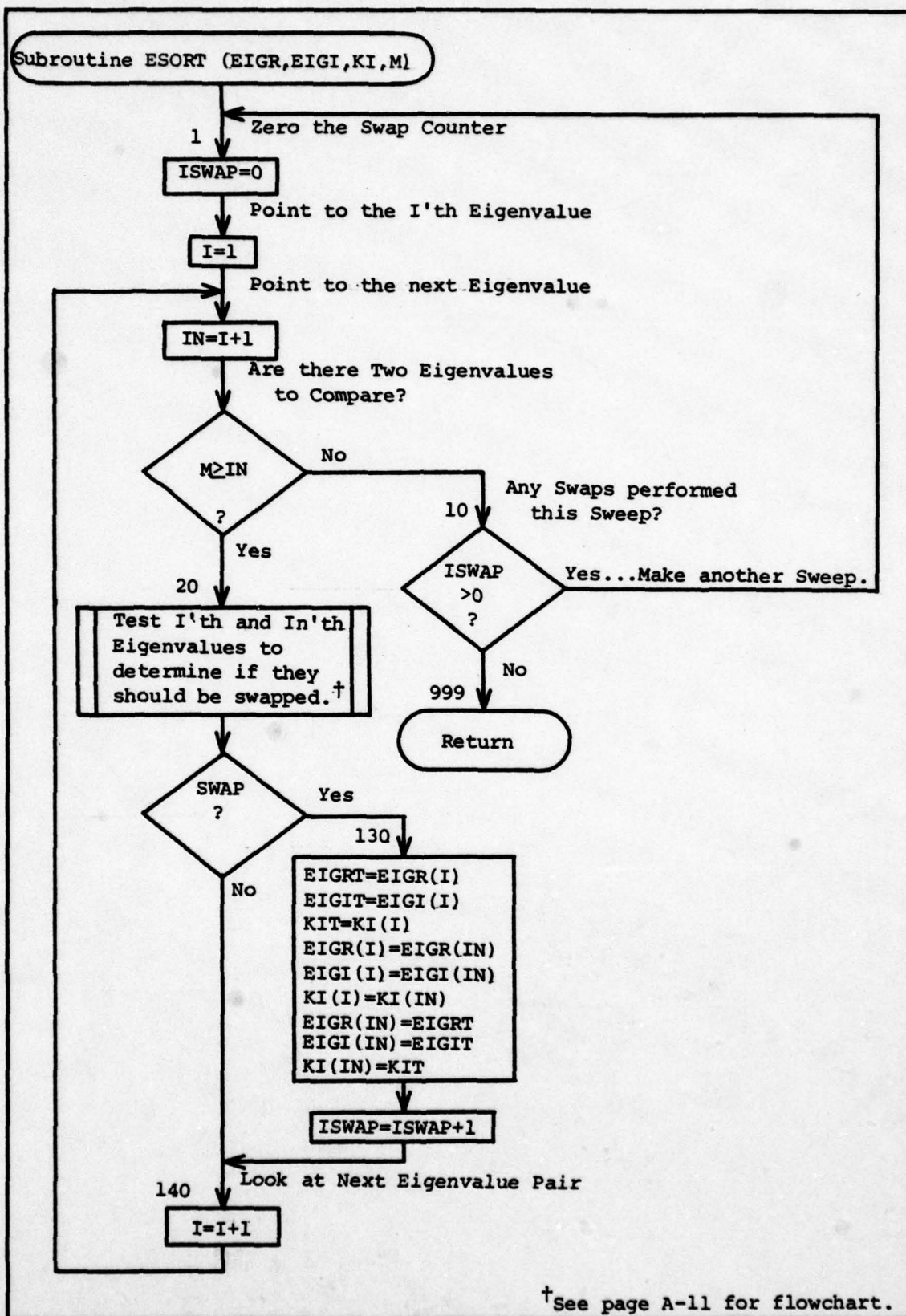


Figure A-3. Flowchart for subroutine ESORT.

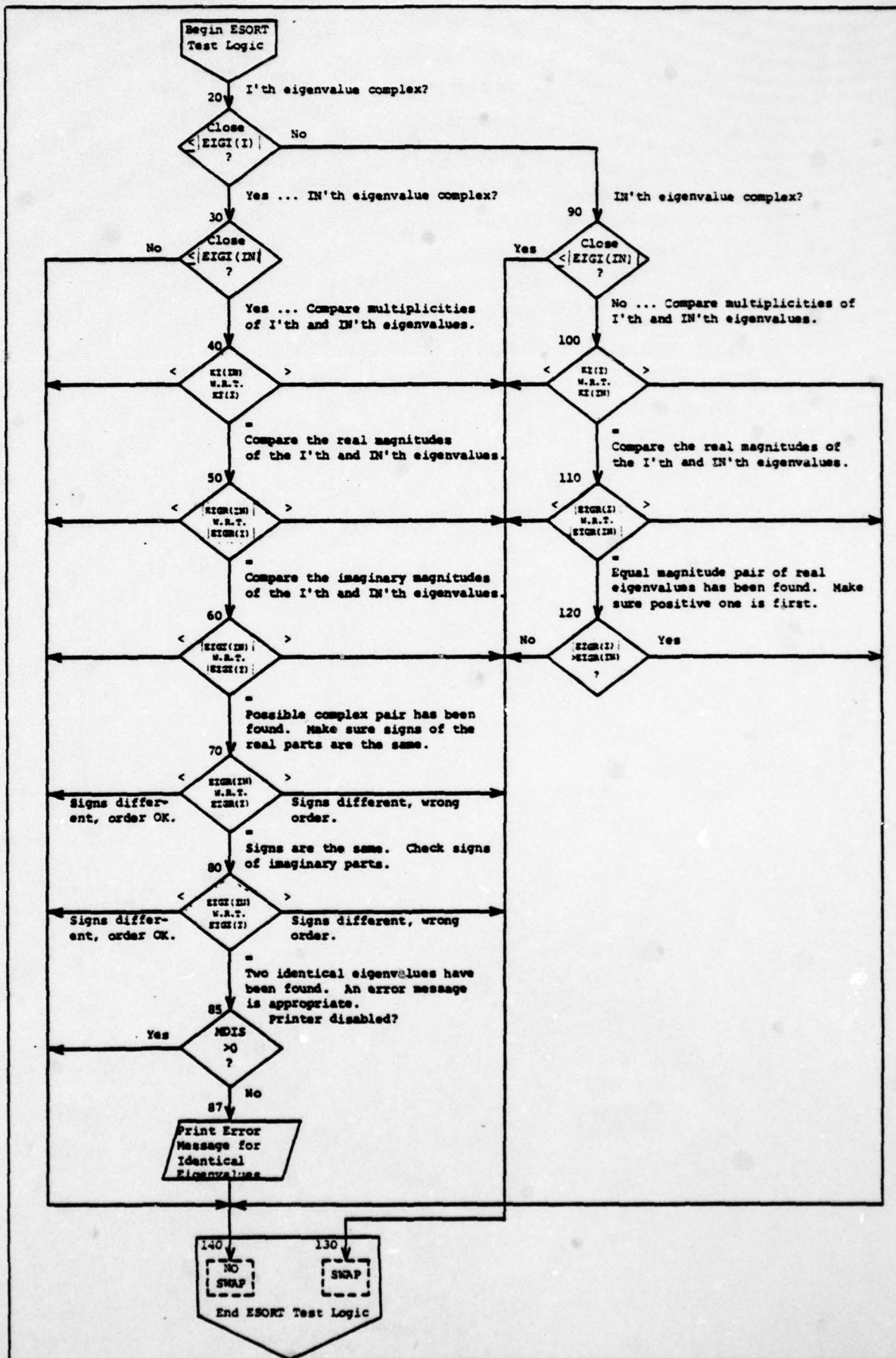


Figure A-4. Flowchart for test logic of subroutine ESORT.

Subroutine EVAL

Subroutine EVAL finds the value of the input polynomial at the specified value of the variable. The input polynomial coefficients and the value of the variable may be complex.

Subroutine statement: SUBROUTINE EVAL (AR,AI,AKR,AKI,EIGR,EIGI,
NS,BR,BI)

Subroutines called: CGAIN

Variables:

| | | |
|-------|---|--|
| AR) | = | Arrays containing the real and imaginary parts |
| AI) | | of the coefficients of the input polynomial |
| AKR) | = | Real and imaginary parts of the gain of the input |
| AKI) | | polynomial |
| EIGR) | = | Real and imaginary parts of the value at which the poly- |
| EIGI) | | nomial is to be evaluated (must be chosen by the |
| | | calling routine) |
| NS | = | Number of storage locations required for the input |
| | | polynomial; NS=N+1, where N is the order of the |
| | | polynomial |
| BR) | = | Real and imaginary parts of the returned value |
| BI) | | |

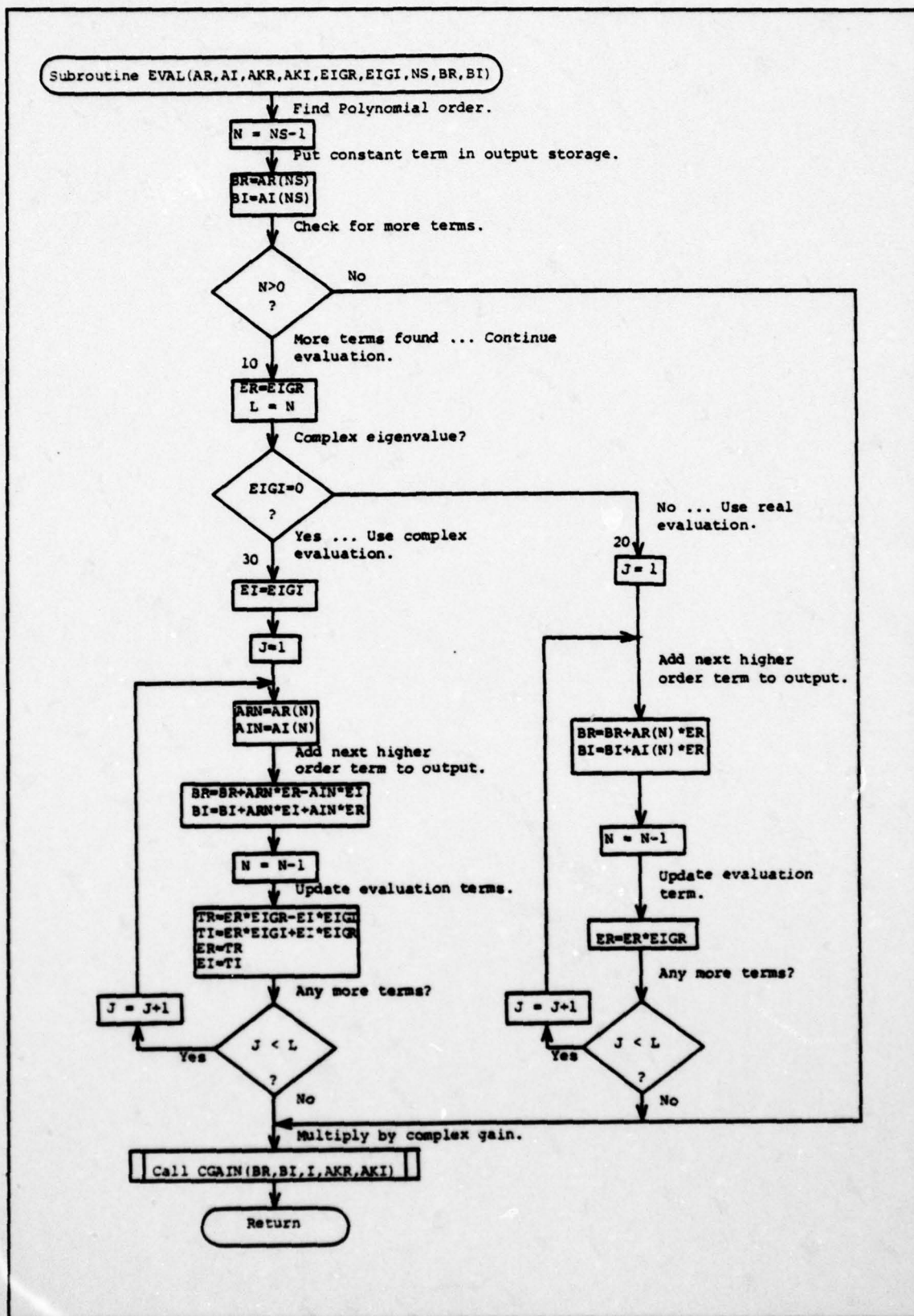


Figure A-5. Flowchart for subroutine EVAL.

Subroutine EXPANDC

Subroutine EXPANDC finds the coefficients for the polynomial expansion of a collection of root factors. EXPANDC reads the roots from a root array and returns the coefficients in a polynomial coefficient array. The coefficient for the highest power of the variable is the first element in the coefficient array. The routine handles both real and complex roots and does not require that complex roots have conjugate pairs. The maximum size of the polynomial found is limited only by the storage allocated for CR and CI in the calling program.

Subroutine statement: SUBROUTINE EXPANDC (ROOTR,ROOTI,NS,BR,BI,CR,CI,
CKR,CKI)

Subroutines called: POLYMC

Variables:

| | | |
|------------------|---|--|
| ROOTR) ROOTI) | = | Arrays containing the real and imaginary parts of the roots whose factors are to be expanded into a polynomial |
| NS | = | The number of storage locations required for the output polynomial; equal to NF+1 where NF is the number of factors |
| BR) BT) | = | NS-dimensioned temporary storage for intermediate results of the polynomial expansion |
| CR) CI) | = | NS-dimensioned arrays containing the real and imaginary parts of the coefficients of the output polynomial |
| CKR) CKI) | = | Real and imaginary parts of the output polynomial gain multiplier; always equal to 1. and 0., respectively |
| CLOSE | = | User specified value used as a null tester. If the magnitude of any value tested is less than CLOSE, that value is set to zero. CLOSE is stored in blank common and must be specified in the main program. |

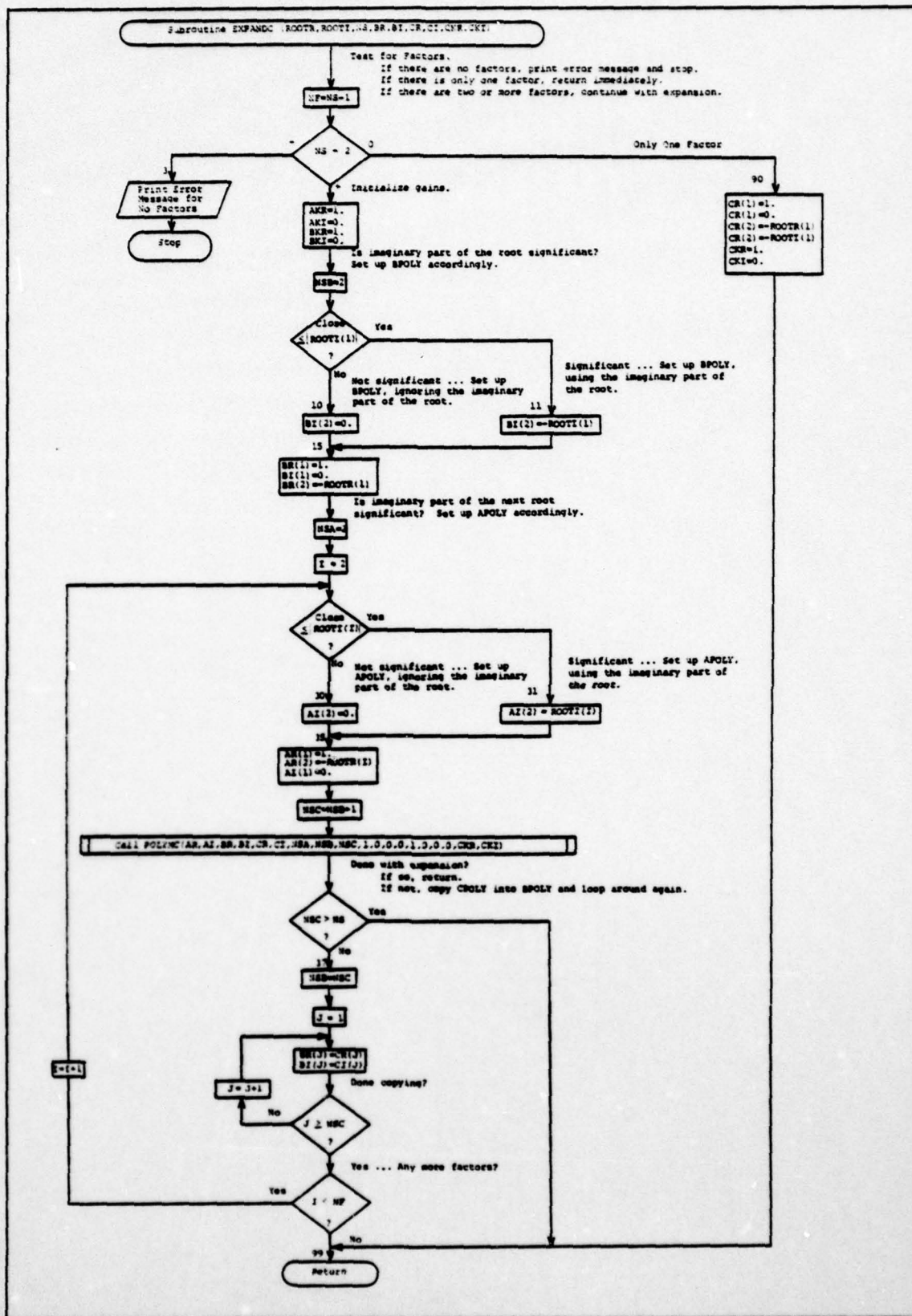


Figure A-6. Flowchart for subroutine EXPANDC.

Subroutine FRACDIF

Subroutine FRACDIF accepts a polynomial fraction and finds its derivative using the quotient rule:

$$\frac{d}{dx} \left(\frac{u}{v} \right) = \frac{u \frac{dv}{dx} - v \frac{du}{dx}}{v^2}$$

The coefficients of the input numerator and/or denominator polynomial may be complex.

Subroutine statement: FRACDIF (PUR,PUI,PVR,PVI,NUS,NVS,PNR,PNI,PDR,PDI,NNS,NDS,NT,UR,UI,VR,VI,RNR,RNI,DR,DI,TEMP1R,TEMP1I,TEMP2R,TEMP2I,TEMP3R,TEMP3I)

Subroutines called: POLYDIF, POLYMC, POLYSUB

Variables:

PUR,PUI) = Arrays containing the real and imaginary parts of the coefficients of the input polynomials, U and V
PVR,PVI)

NUS) = Number of storage locations needed to store the coefficients of the input polynomials
NVS)

PNR,PNI) = Arrays containing the real and imaginary parts of the coefficients of the output numerator polynomial (PNx) and denominator polynomial (PDx)
PDR,PDI)

NNS) = Number of storage locations needed to store the coefficients of the output numerator and denominator polynomials, where:
NDS)

$$NNS = NUS + NVS$$

$$NDS = 2NVS - 1$$

NT = Number of temporary storage locations needed for the numerator calculations

$$NT = NON(L) + NOD(I)$$

where: NON(L) is storage required for Lth numerator polynomial
NOD(I) is storage required for Ith denominator polynomial

UR,UI)
VR,VI) = Real and imaginary parts of the gains associated with
RNR,RNI) polynomials U, V, PNx, PDx
DR,DI)

TEMP1R,TEMP1I)
TEMP2R,TEMP2I) = Temporary storage locations for numerator calculations
TEMP3R,TEMP3I)

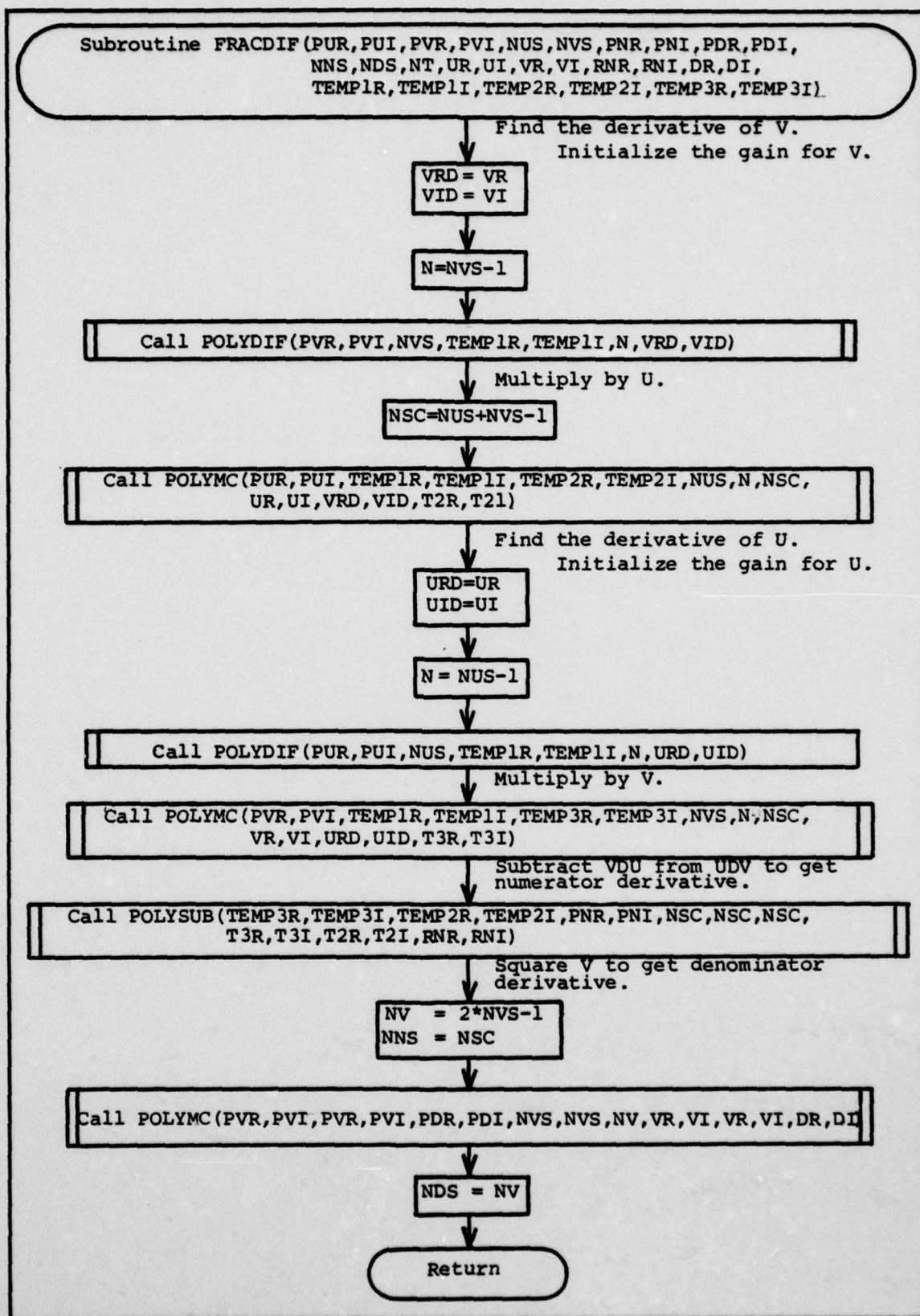


Figure A-7. Flowchart for subroutine FRACDIF.

Function NFACT

Function NFACT finds the factorial value of the argument, (IF), entered. Negative arguments cause an error message to be printed and cause the program to terminate.

Function statement: FUNCTION NFACT(IF)

Subroutine called: None

Variables:

IF = The integer for which the factorial value is desired.

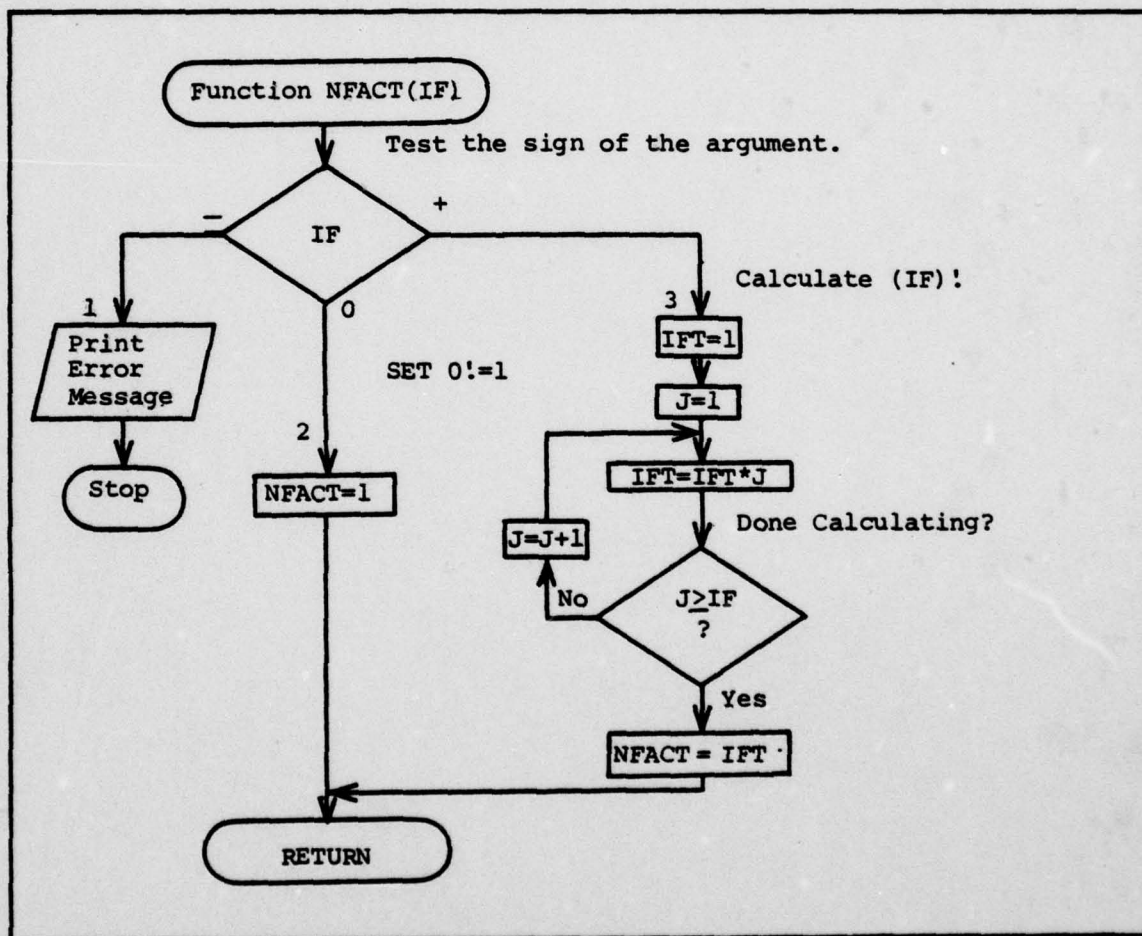


Figure A-8. Flowchart for function subprogram NFACT.

Subroutine OUT2C

Subroutine OUT2C prints out the elements of a two-dimensional array and labels each element with the array name and its position in the array. The real and imaginary parts of the complex array are treated separately.

Subroutine statement: SUBROUTINE OUT2C (NR,NC,AR,AI,IDR,IDI)

Subroutines called: None

Variables:

NR = Number of rows to be printed out
NC = Number of columns to be printed out
AR) = Arrays containing the real and imaginary parts of
AI) = the array to be printed
IDR) = Hollerith labels for the real and imaginary parts of each
IDI) = element printed; these must be specified in Hollerith
format by the calling program

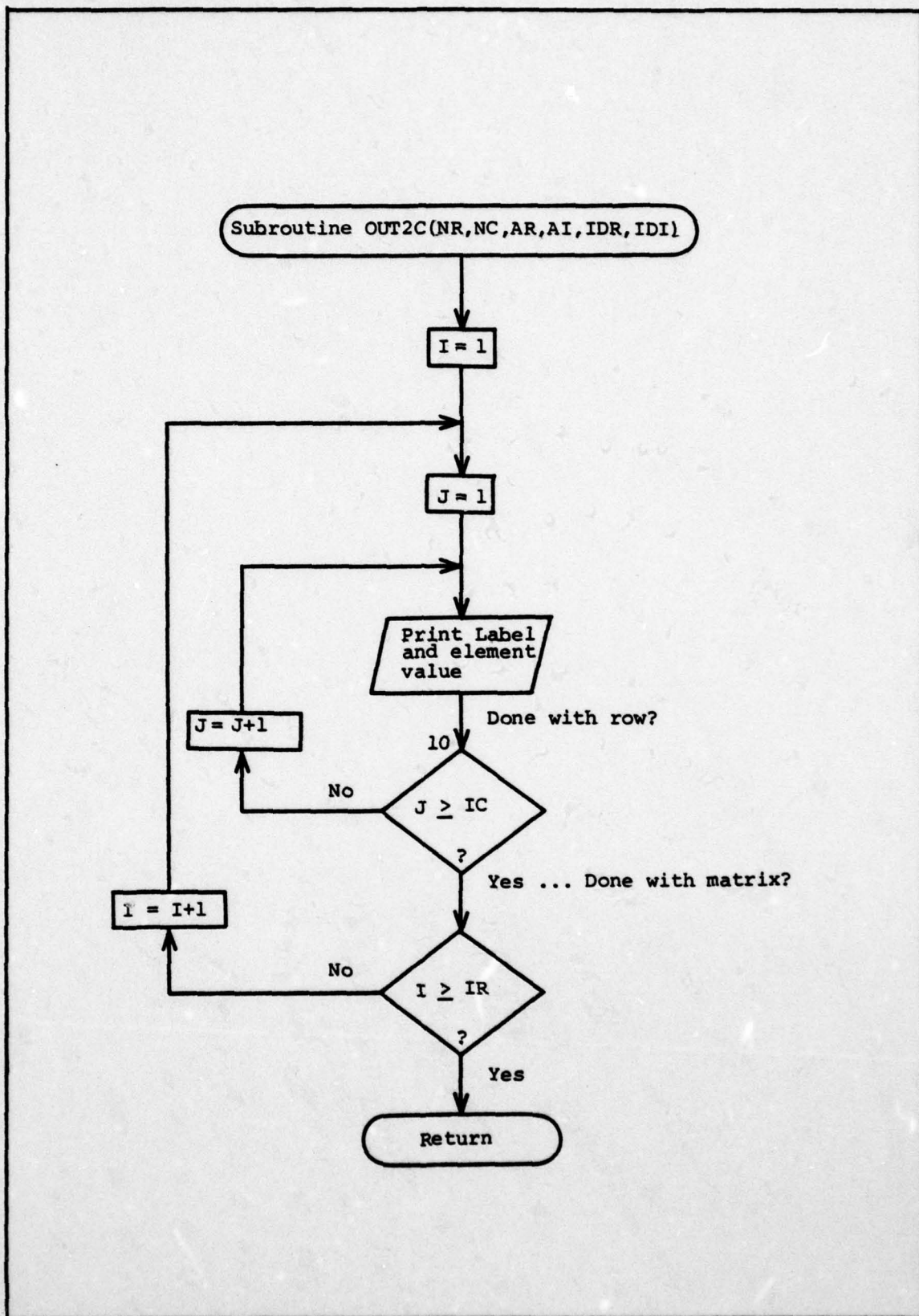


Figure A-9. Flowchart for subroutine OUT2C.

Subroutine POLYADD

Subroutine POLYADD adds two polynomials, POLYA and POLYB, and stores the result in POLYC. The coefficients of the polynomials may be complex. After the addition is complete, the first coefficient of the output is tested to determine if it is unity. If it is not, subroutine UNITY is called to unitize it.

Subroutine statement: SUBROUTINE POLYADD(AR,AI,BR,BI,CR,CI,NSA,NSB,NSC,
AKR,AKI,BKR,BKI,CKR,CKI)

Subroutines called: CGAIN, UNITY

Variables:

AR)
AI)
BR) = Arrays containing the real and imaginary parts of
BI) the coefficients of POLYA, POLYB, and POLYC
CR)
CI)

NSA)
NSB) = Number of storage locations required for the coefficients
NSC) of POLYA, POLYB, and POLYC

AKR)
AKI)
BKR) = Real and imaginary parts of the gain terms for
BKI) POLYA, POLYB, and POLYC
CKR)
CKI)

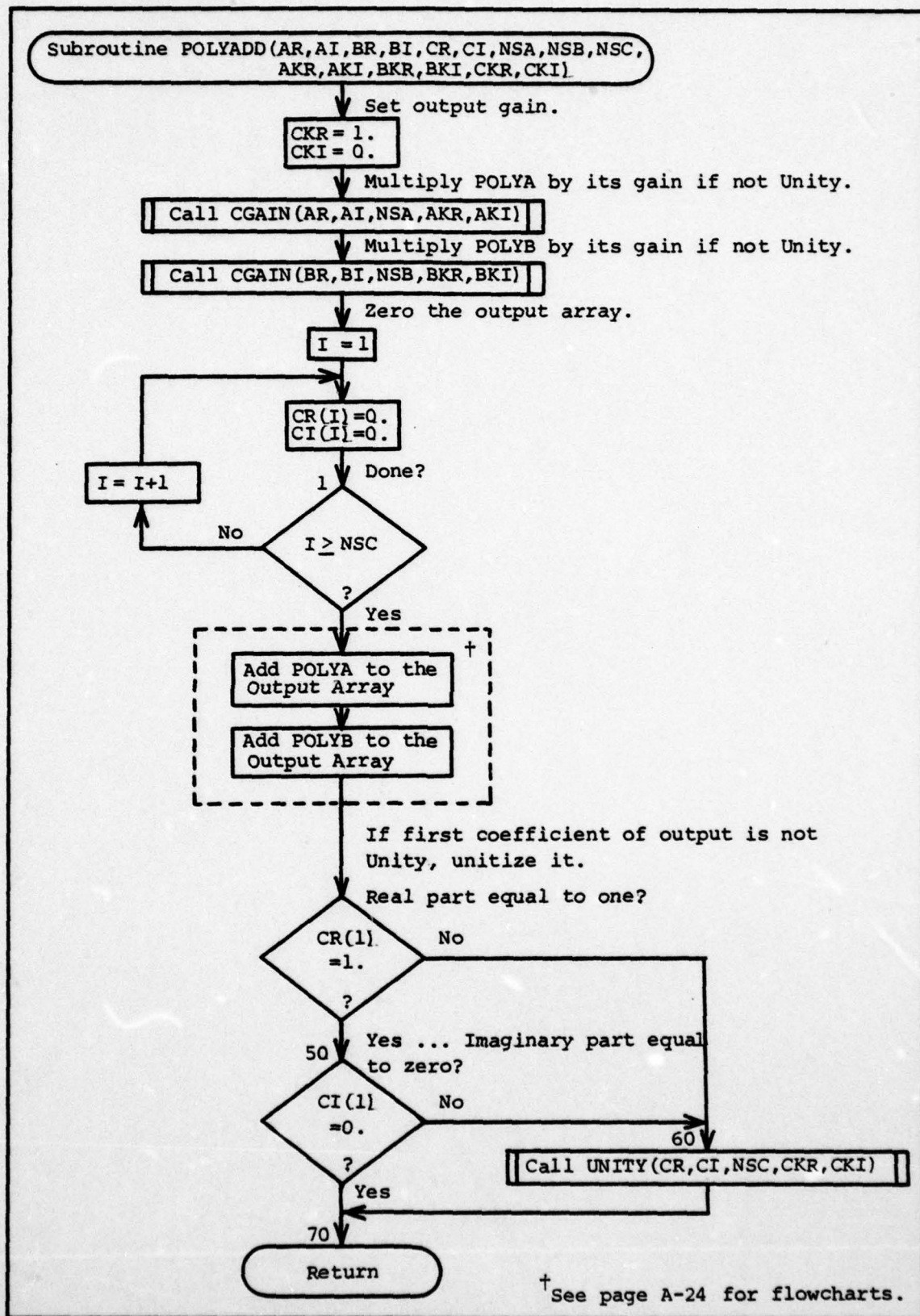


Figure A-10. Flowchart for subroutine POLYADD.

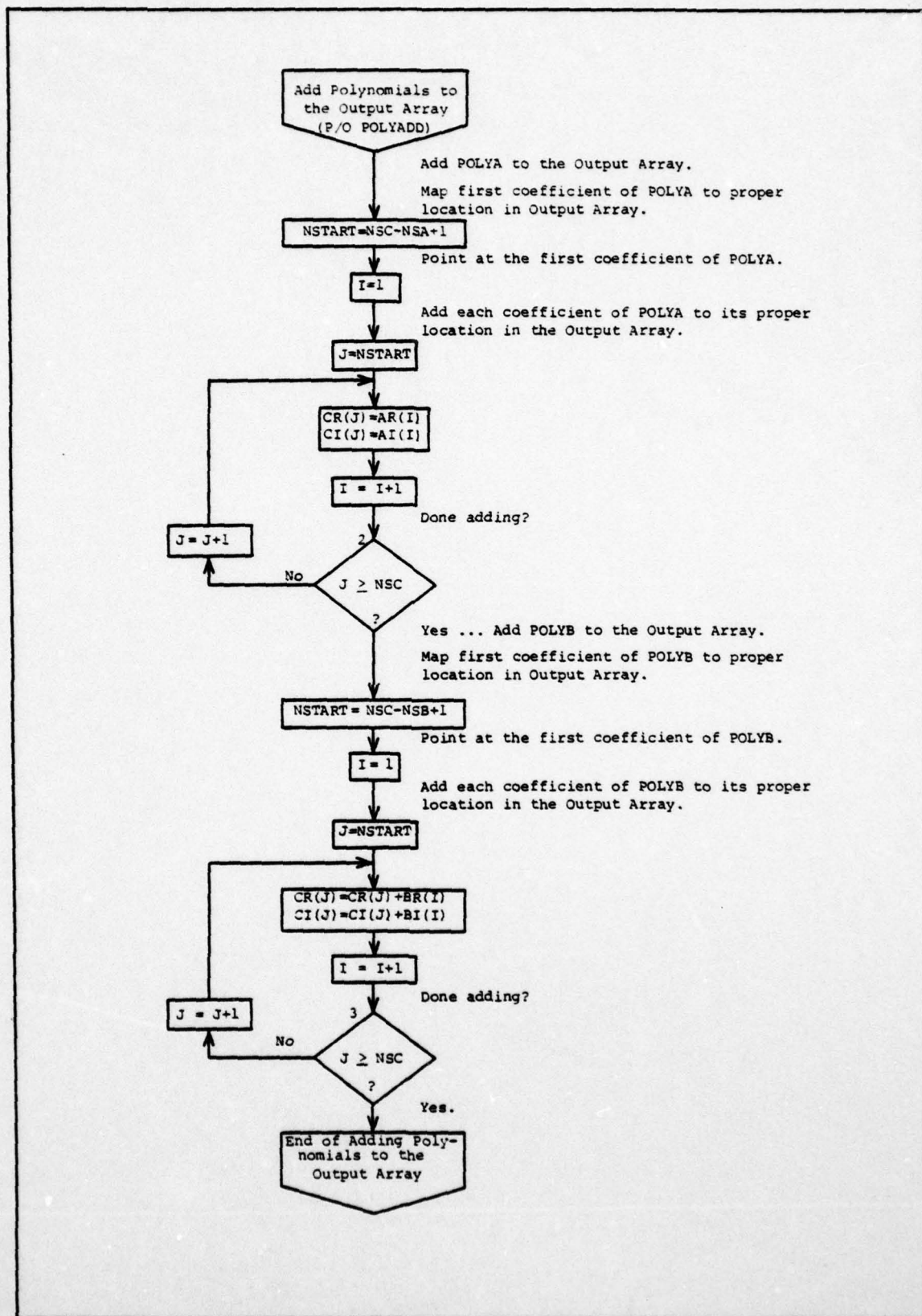


Figure A-11. Flowchart for addition steps of subroutine POLYADD.

Subroutine POLYDIF

Subroutine POLYDIF accepts a polynomial input and returns the derivative. The coefficients of the input polynomial may be complex.

Subroutine statement: SUBROUTINE POLYDIF(POLYR,POLYI,NS,DPOLYR,
DPOLYI,N,PKR,PKI)

Subroutines called: CGAIN

Variables:

POLYR) = Arrays containing the real and imaginary parts of the
POLYI) = coefficients of the input polynomial

DPOLYR) = Arrays containing the real and imaginary parts of the
DPOLYI) = coefficients of the output (derivative) polynomial

NS) = Number of storage locations required for the
N) = coefficients of POLY and DPOLY (Note that:
NS = NI+1, N = NI; where NI is the order of the
input polynomial)

PKR) = Real and imaginary parts of the polynomial gain
PKI)

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 12/1
INVESTIGATION OF INVERSE VANDERMONDE MATRIX CALCULATION FOR LIN--ETC(U)
MAR 79 D P SEYLER

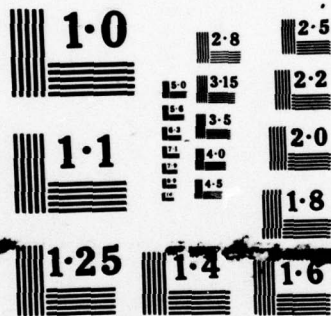
AFIT/GGC/EE/79-2

NL

2 OF 2
AD
A089241

100

END
DATE
FILMED
7-79
DOC



NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

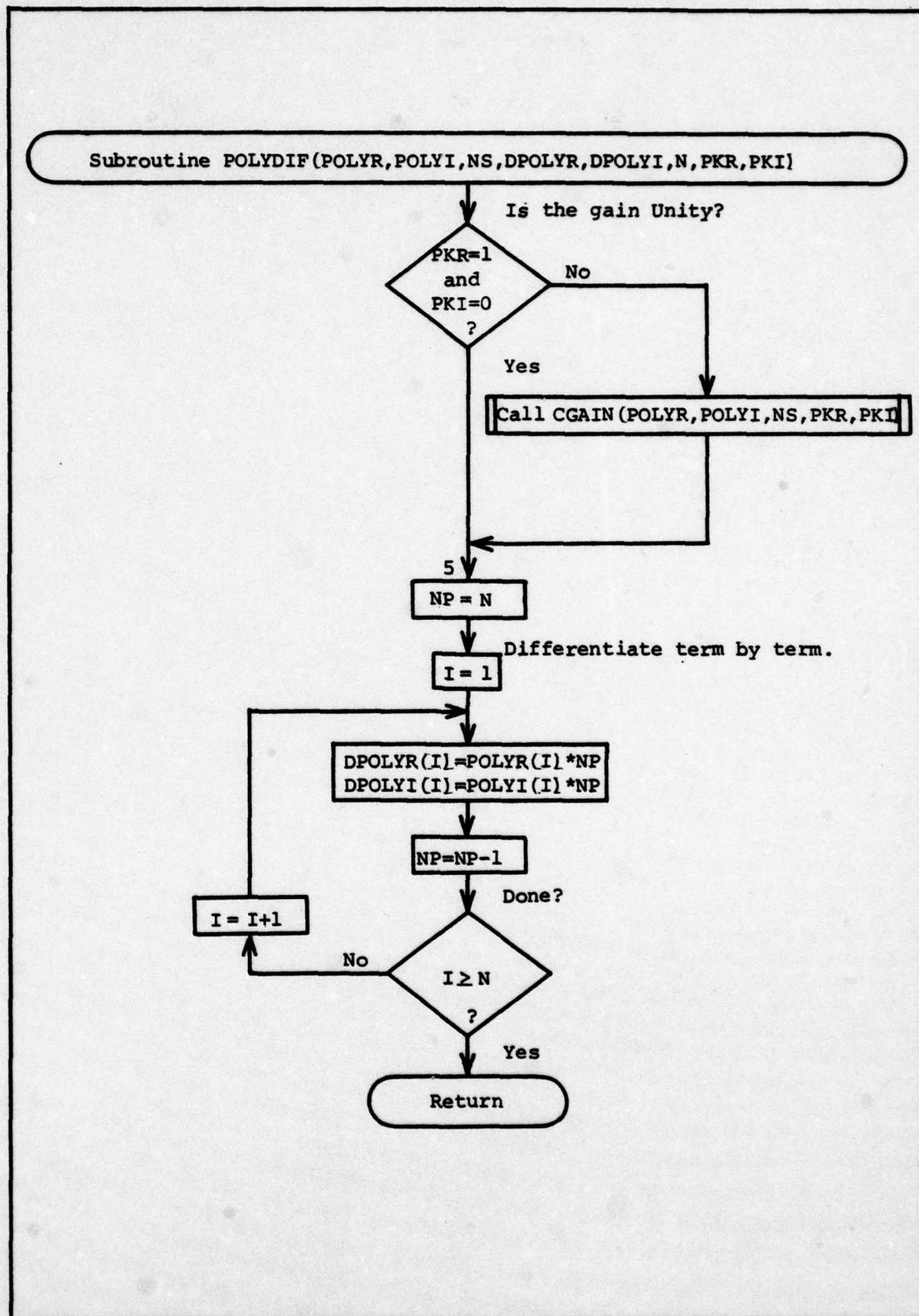


Figure A-12. Flowchart for subroutine POLYDIF.

Subroutine POLYMC

Subroutine POLYMC finds the coefficients of the product of two polynomials. The routine handles all combinations of real and complex input coefficients. The order of the polynomials involved is limited only by the storage allocated for them by the calling program.

Subroutine statement: SUBROUTINE POLYMC (AR, AI, BR, BI, CR, CI, NSA, NSB, NSC, AKR, AKI, BKR, BKI, CKB, CKI)

Subroutines called: UNITY

Variables:

AR) = Arrays containing the real and imaginary parts of the coefficients of the first polynomial multiplicand, POLYA
AI) =

BR) = Arrays containing the real and imaginary parts of the coefficients of the second polynomial multiplicand, POLYB
BI) =

CR) = Arrays containing the real and imaginary parts of the coefficients of the polynomial product, POLYC
CI) =

NSA) = Number of storage locations required for the coefficients of POLYA, POLYB, POLYC; each is one integer larger than the corresponding polynomial order (Note: NSC = NSA+NSB-1)
NSB) =
NSC) =

AKR) = Real and imaginary parts of the gains associated with POLYA, POLYB, POLYC
AKI) =
BKR) =
BKI) =
CKR) =
CKI) =

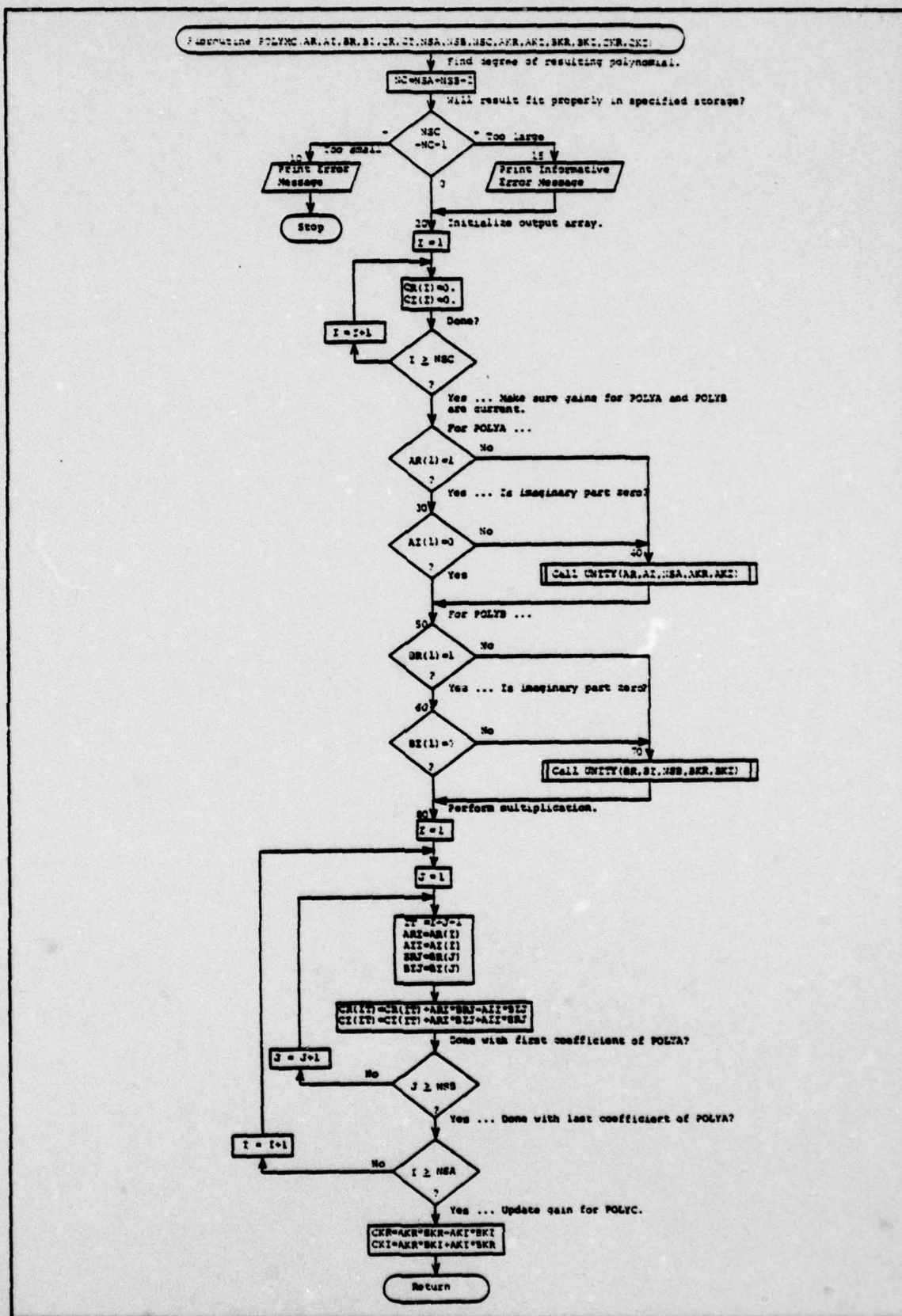


Figure A-13. Flowchart for subroutine POLYMC.

Subroutine POLYSUB

Subroutine POLYSUB subtracts POLYB from POLYA by changing the sign of the gain of POLYB and calling subroutine POLYADD. The coefficients of the polynomials may be complex.

Subroutine statement: SUBROUTINE POLYSUB (AR,AI,BR,BI,CR,CI,NSA,NSB,NSC,
AKR,AKI,BKR,BKI,CKR,CKI)

Subroutines called: POLYADD

Variables:

AR)
AI)
BR) = Arrays containing the real and imaginary parts of the
BI) = coefficients of the polynomials operated upon according
CR) to $POLYC = POLYA - POLYB$
CI)

NSA) Number of storage locations required for the coefficients
NSB) = of POLYA, POLYB, and POLYC, respectively. Note that NSC
NSC) equals the larger of the two values NSA and NSB.

AKR)
AKI)
BKR) = Real and imaginary parts of the gain multipliers for
BKI) = POLYA, POLYB, and POLYC, respectively.
CKR)
CKI)

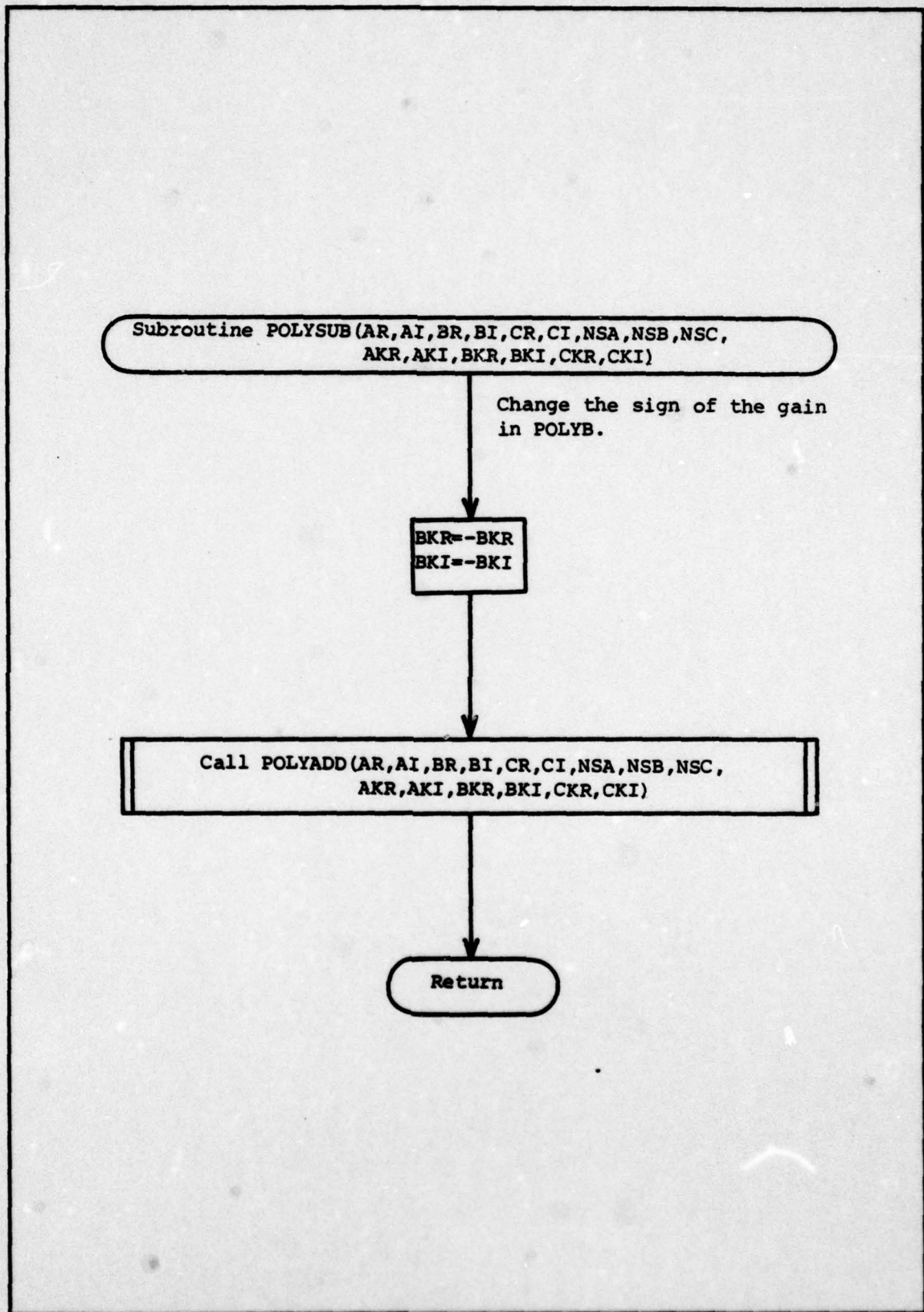


Figure A-14. Flowchart for subroutine POLYSUB.

Subroutine RAT

Subroutine RAT rationalizes a complex fraction by multiplying and dividing the fraction by the complex conjugate of the denominator. The denominator coefficients are tested for nullity prior to the last step of rationalization, and if they are null, an error message is printed and execution is terminated.

Subroutine statement: SUBROUTINE RAT(RNR,RNI,DR,DI,RR,RI)

Subroutines called: None

Variables:

RNR) = Real and imaginary parts of the input numerator
RNI)

DR) = Real and imaginary parts of the input denominator
DI)

RR) = Real and imaginary parts of the rationalized output
RI)

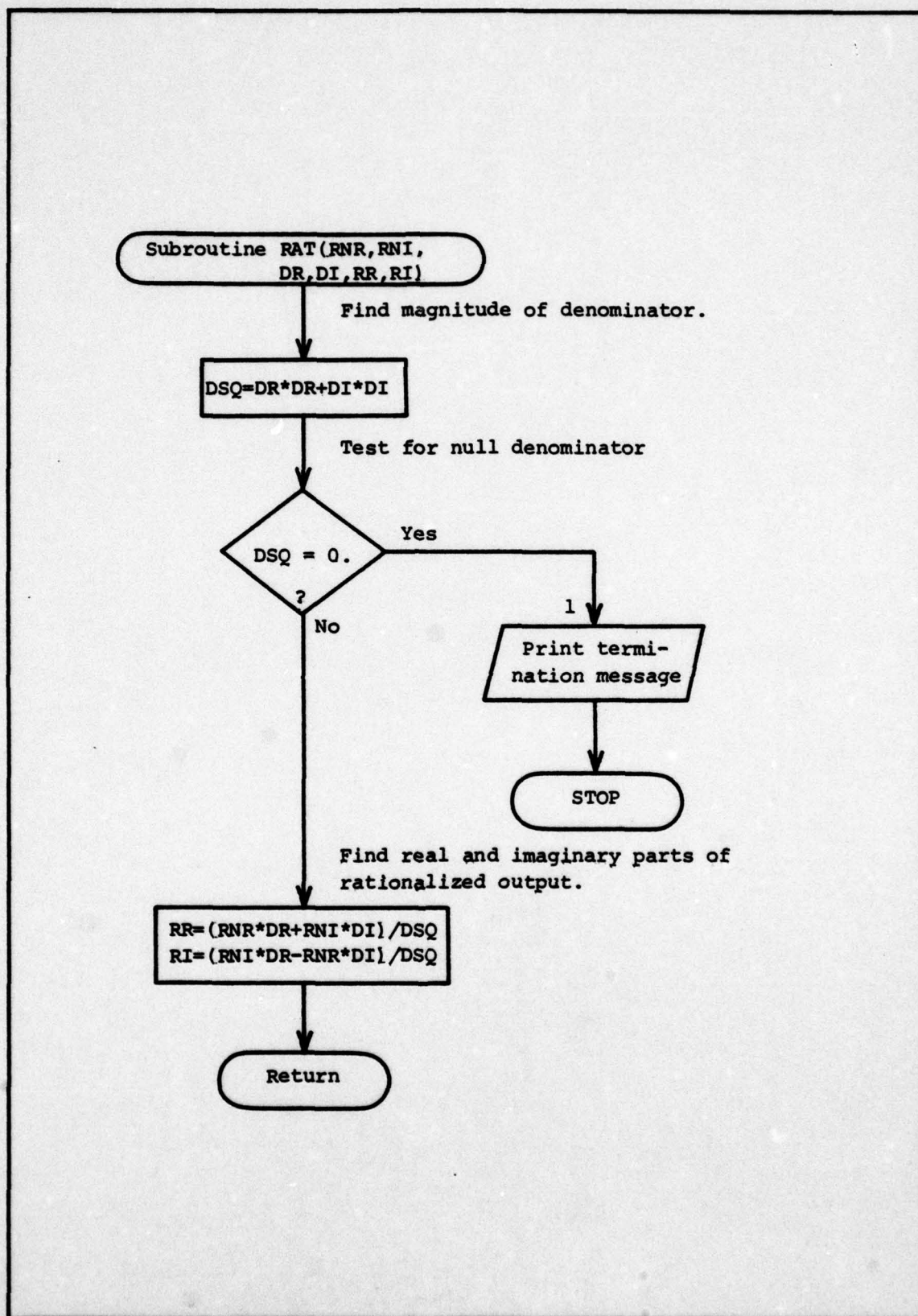


Figure A-15. Flowchart for subroutine RAT.

Subroutine ROOTAY

Subroutine ROOTAY expands a collection of eigenvalues with their associated multiplicities into an array containing one root for each occurrence of each eigenvalue. Both real and complex eigenvalues may be used. The routine forms the root array beginning with the IB'th eigenvalue in the input array and uses all the remaining eigenvalues in the array. Thus the input array must have the eigenvalues arranged so that any eigenvalues to be excluded from the expansion precede the IB'th eigenvalue.

Subroutine statement: SUBROUTINE ROOTAY(EIGR,EIGI,K,M,IS,IB,
ROOTR,ROOTI)

Subroutines called: None

Variables:

EIGR) = Arrays containing the real and imaginary parts of the
EIGI) = input collection of eigenvalues

K = An array containing the multiplicities of the eigenvalues

M = The number of different eigenvalues

IS = The number of storage locations needed to contain the
resulting array of roots;

note that $IS = \sum_{i=1}^M K(i)$, the summation of the eigenvalue
multiplicities

ROOTR) = Arrays containing the real and imaginary parts of the
ROOTI) = roots in the output array

IB = The number of the eigenvalue with which to begin forming
the root array.

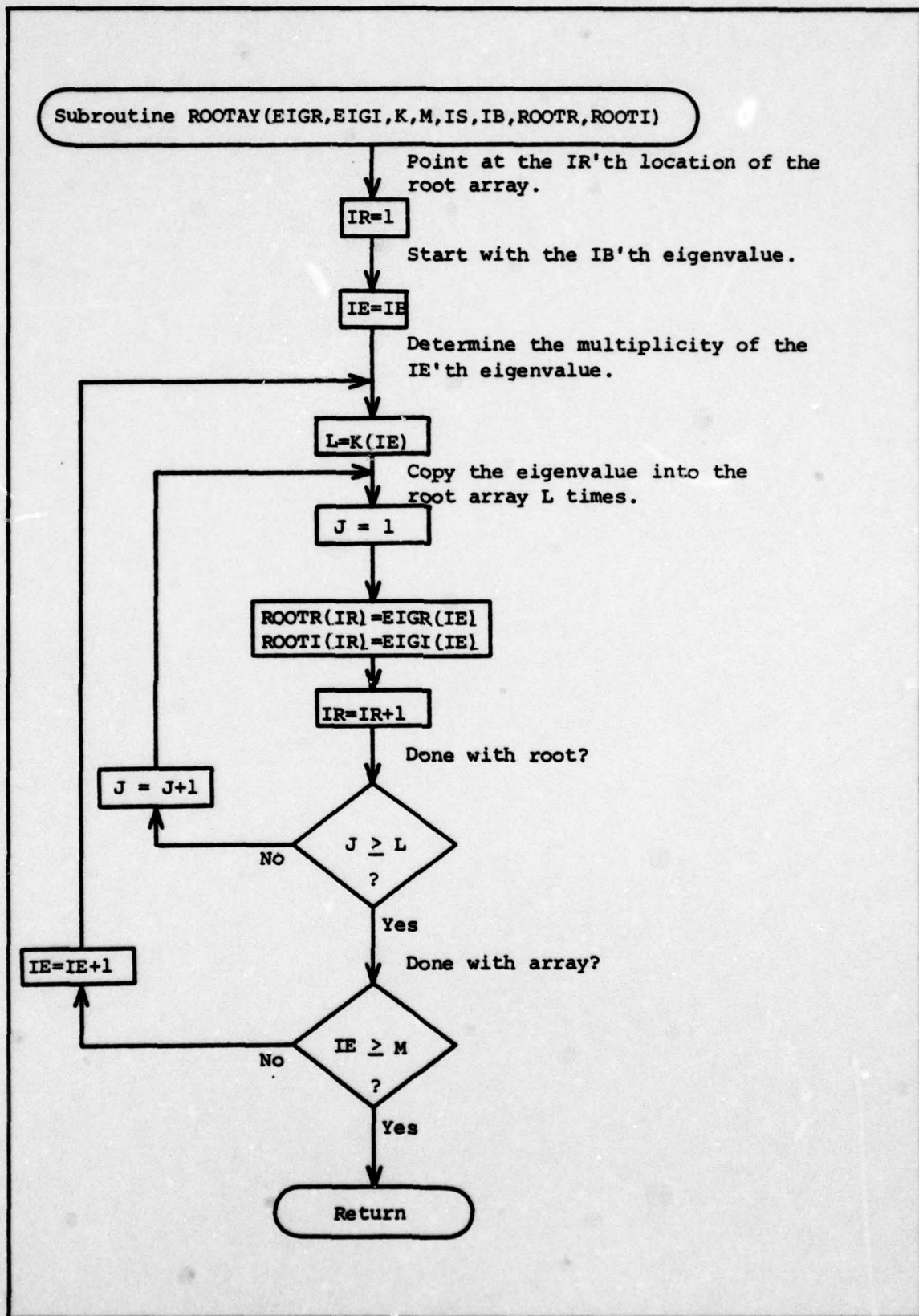


Figure A-16. Flowchart for subroutine ROOTAY.

Subroutine UNITY

Subroutine UNITY operates on the input polynomial to unitize the coefficient of the highest order variable. This is accomplished by extracting the coefficient of the highest order term as a gain multiplier. The polynomial gain is then updated to reflect the extraction. The input coefficients may be real or complex. In the event that the first coefficient of the input polynomial is zero, it is ignored and the next one is tested. If all the coefficients are zero, the gain and constant term are set to zero, and a warning message is printed if the printer has not been disabled.

Subroutine statement: SUBROUTINE UNITY(AR,AI,NS,AKR,AKI)

Subroutines called: None

Variables:

- AR) = Arrays containing the real and imaginary parts of the
- AI) = coefficients of the input polynomial
- NS = Number of storage locations required for the coefficients of the input polynomial
- AKR) = Real and imaginary parts of the polynomial gain
- AKI)
- CLOSE = User specified magnitude used in testing the significance of variables. If the magnitude of the tested variable is less than CLOSE, it is assumed to be zero. CLOSE is stored in blank common.
- MDIS = User specified printer disable flag. If MDIS is greater than zero all print steps using the flag are disabled. MDIS is stored in blank common.

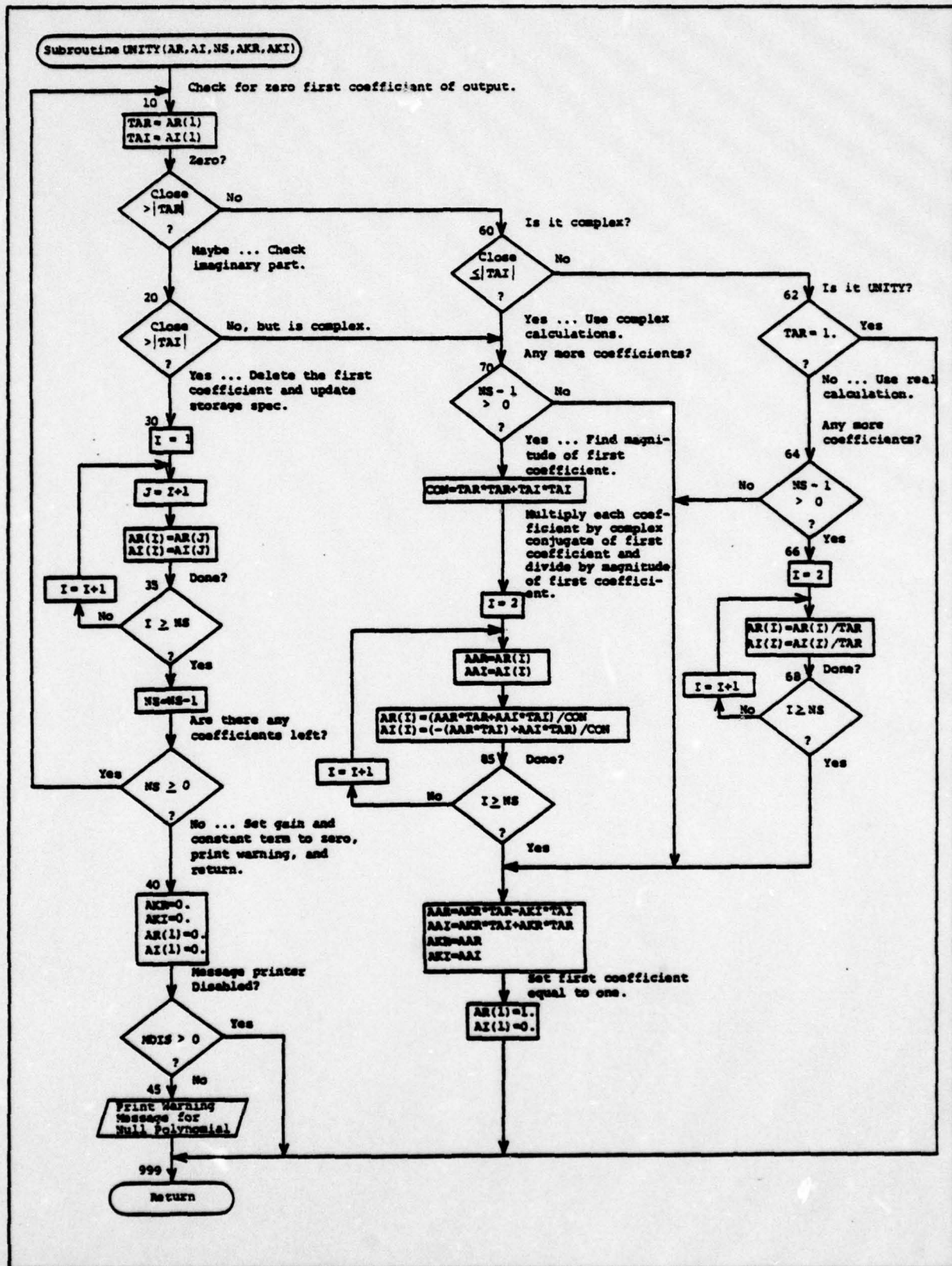


Figure A-17. Flowchart for subroutine UNITY.

Subroutine VANINV

Subroutine VANINV is an implementation of Csáki's generalized algorithm for finding the inverse of the Vandermonde matrix. It accepts an ordered array of real and complex eigenvalues and their multiplicities, and returns the inverse Vandermonde matrix. Prior to calling VANINV, the eigenvalues must be arranged in the order described for subroutine ESORT.

Subroutine statement: SUBROUTINE VANINV(EIGR,EIGI,KI,N,M,NS,MS,WR,WI,
NW1,NW2,AR,AI,FR,FI,NR,NI,DR,DI,GR,GI,NOD,NON,
PNR,PNI,PDR,PDI,PUR,PUI,PVR,PVI,T1R,T1I,T2R,T2I,
T3R,T3I,DKR,DKI)

Subroutines called: COPYPOL,EVAL,EXPANDC,FRACDIF,RAT,ROOTAY

Function subprograms called: NFACT

Variables:

1) Formal parameters

EIGR) = Arrays containing the real and imaginary parts of the M
EIGI) = different eigenvalues of the N'th order system

KI = Array containing the multiplicities corresponding to the
system eigenvalues

N = System order (note that $N = \sum_{i=1}^M K(i)$)

M = Number of different eigenvalues (complex conjugates are
considered to be different)

NS) = Storage size parameters, equal to N+1 and M+1,
MS) = respectively

WR) = N x N arrays into which the real and imaginary parts of
WI) = the inverse Vandermonde matrix are placed by VANINV

NW1) = Storage size parameters, equal to
NW2) = $(2^{**}(KI_{\max}-1)) * (N-KI_{\max}) + 1$ and 2N, respectively

AR) = N-dimensioned arrays used for intermediate polynomial
AI) = calculations (short form of POLYA)

FR)
 FI) = NS-dimensional arrays used for intermediate polynomial
 GR) calculations (short forms for POLYF and POLYG)
 GI)
 NR) = (N × N)-dimensioned arrays used to store the real and
 NI) imaginary parts of the N truncated numerator polynomials
 DR) = (M × MS)-dimensioned arrays used to store the real and
 DI) imaginary parts of the M denominator polynomials
 NOD = M-dimensional array used to store the polynomial order
 associated with each of the M denominator polynomials
 NON = N-dimensional array used to store the polynomial order
 associated with each of the N numerator polynomials

PNR,PNI) Working storage for polynomials used in differentiating
 PDR,PDI) = a polynomial fraction by means of the quotient rule:
 PUR,PUI)
 PVR,PVI)

$$\frac{d}{dx} \left(\frac{u}{v} \right) = \frac{u \frac{dv}{dx} - v \left(\frac{du}{dx} \right)}{v^2} = \frac{\text{POLYN}}{\text{POLYD}}$$

Note that all of these must be dimensioned at least NW1

T1R,T1I)
 T2R,T2I) = NW2-dimensional arrays used for temporary storage
 T3R,T3I)

AKR,AKI)
 FKR,FKI)
 GKR,GKI) Real and imaginary parts of the gains associated with
 UR,UI) POLYA, POLYF, POLYG, POLYU, POLYV, POLYN, POLYD
 VR,VI)
 RNR,RNI)
 RDR,RDI)

GNLR) = Real and imaginary parts of the gains associated with
 GNLI) each of the M denominator polynomials. Since in all
 cases: GNLR=1. and GNLI=0., they are defined with a
 data statement.

NDIV = Number of derivatives that must be taken of the current
 inverse Vandermonde element, W(J,L), before evaluation.

NDIVS = Storage for NDIV so initial value can be recalled

RC = Row coefficient for J'th row of I'th block of the inverse
 Vandermonde matrix

Note that $RC = \frac{1.}{(KI(I) - J)!} \frac{1.}{(NDIVS)!}$

NUS)
NVS)
NNS) = Dummy variables used to pass between subroutines the cur-
NDS) rent size of the storage locations of POLYN and POLYD
NT)

RNUMR,RNUMI) = Real and imaginary parts of the polynomial fraction for an
DENR,DENI) element of the inverse Vandermonde prior to rationalization

ZR) = Real and imaginary parts of an element of the inverse
ZI) Vandermonde matrix after rationalization of the fraction
but prior to multiplication by the row coefficient

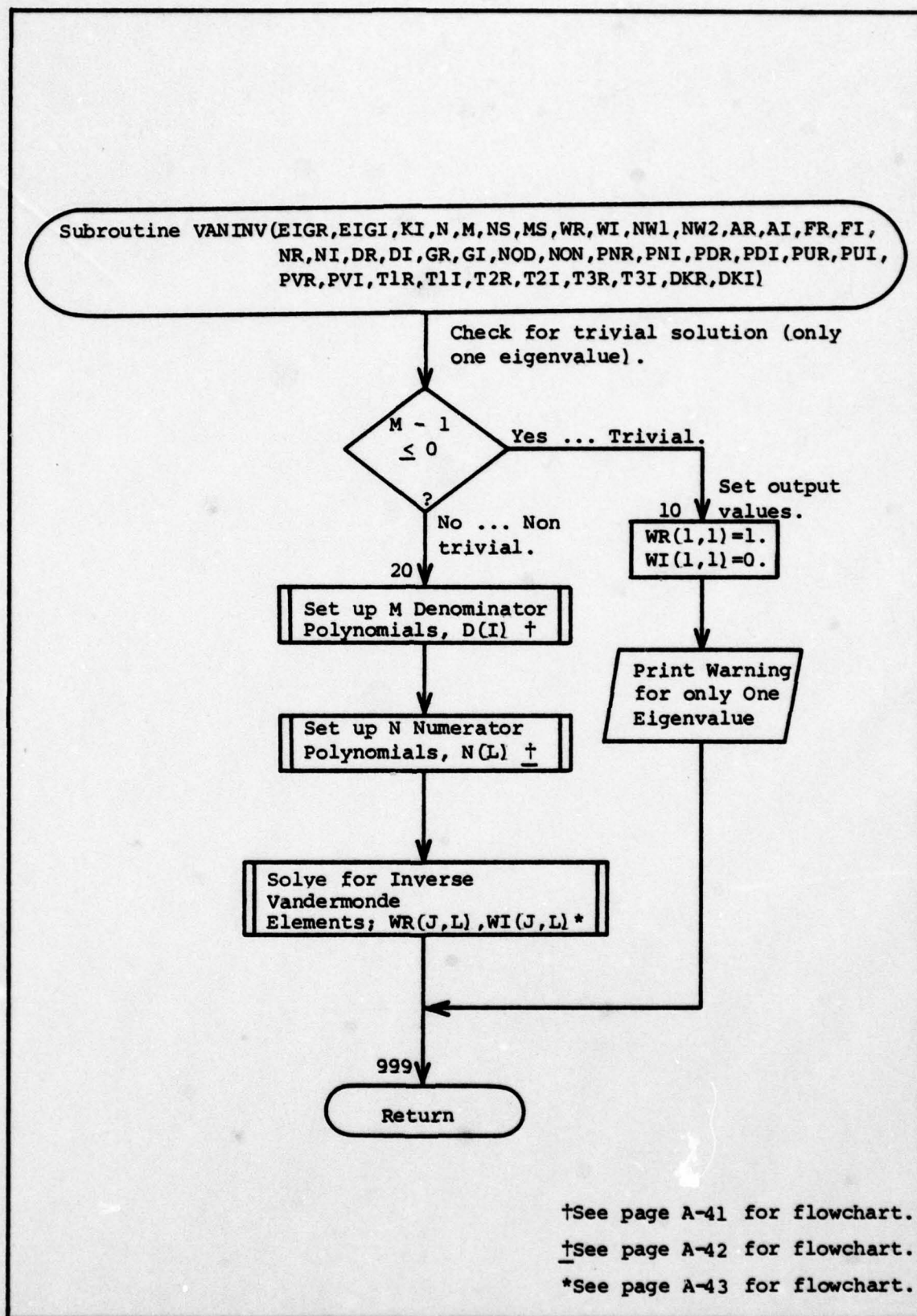


Figure A-18. Flowchart for subroutine VANINV.

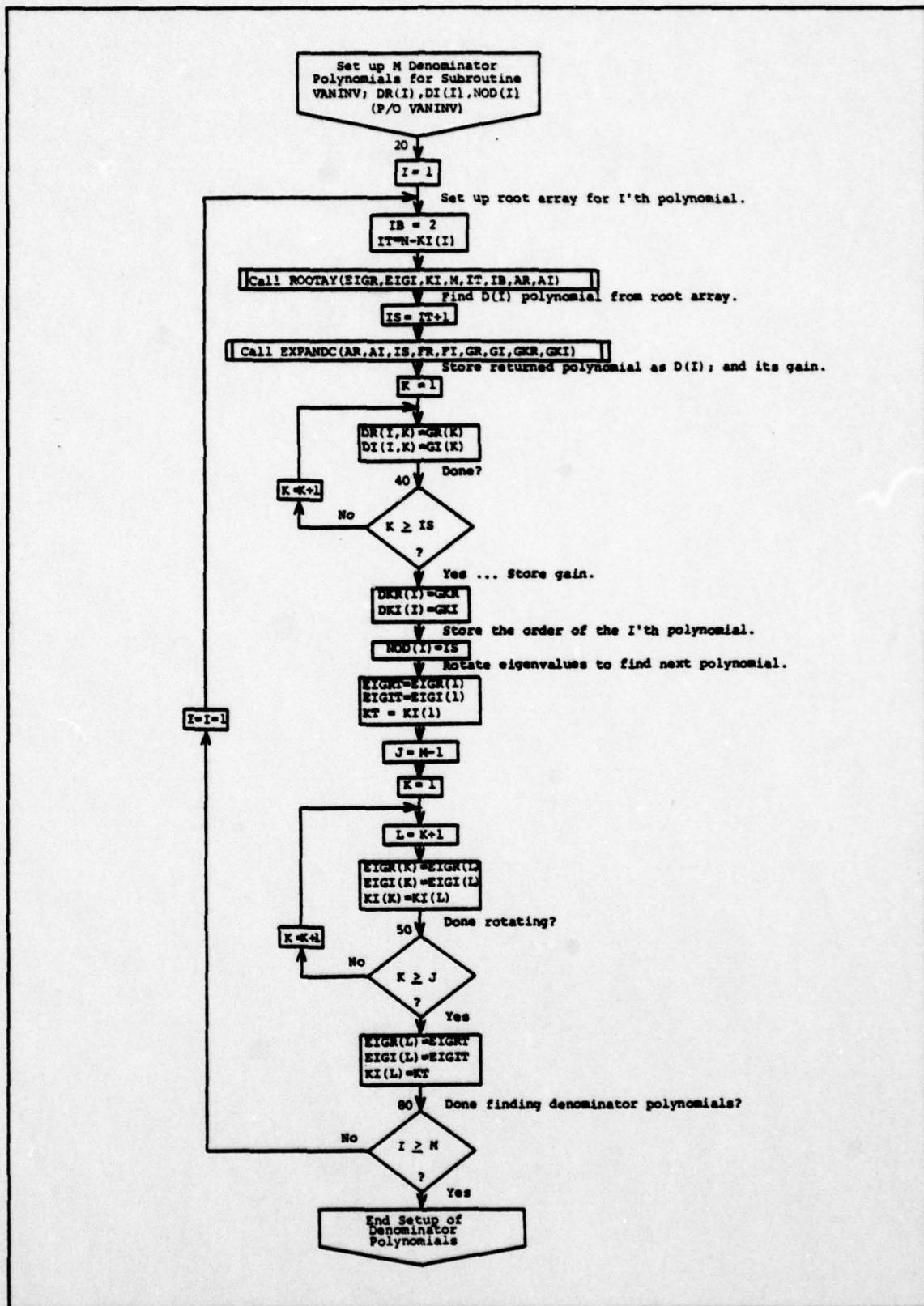


Figure A-19. Flowchart for denominator polynomial setup steps for subroutine VANINV.

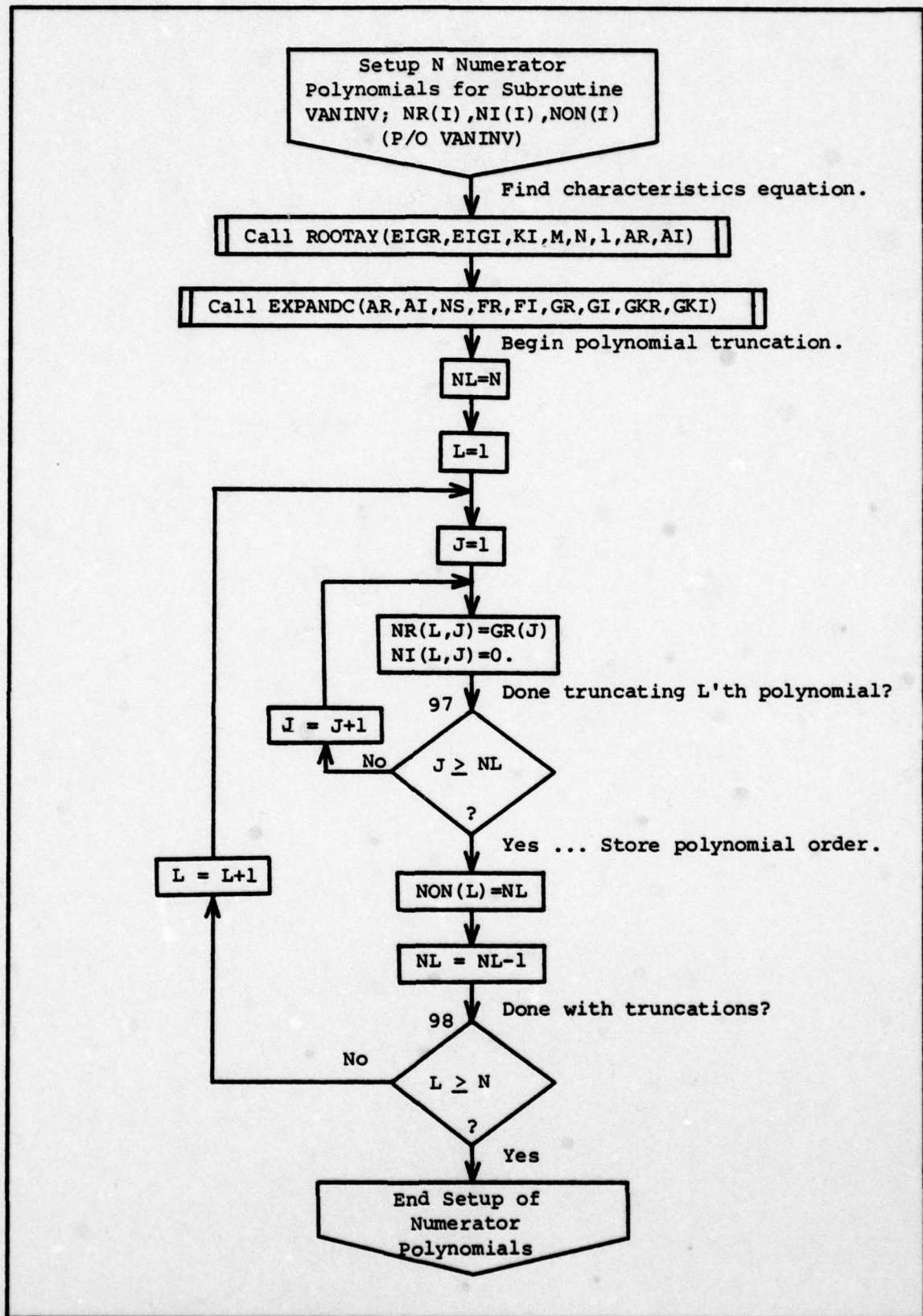


Figure A-20. Flowchart of numerator polynomial setup steps for subroutine VANINV.

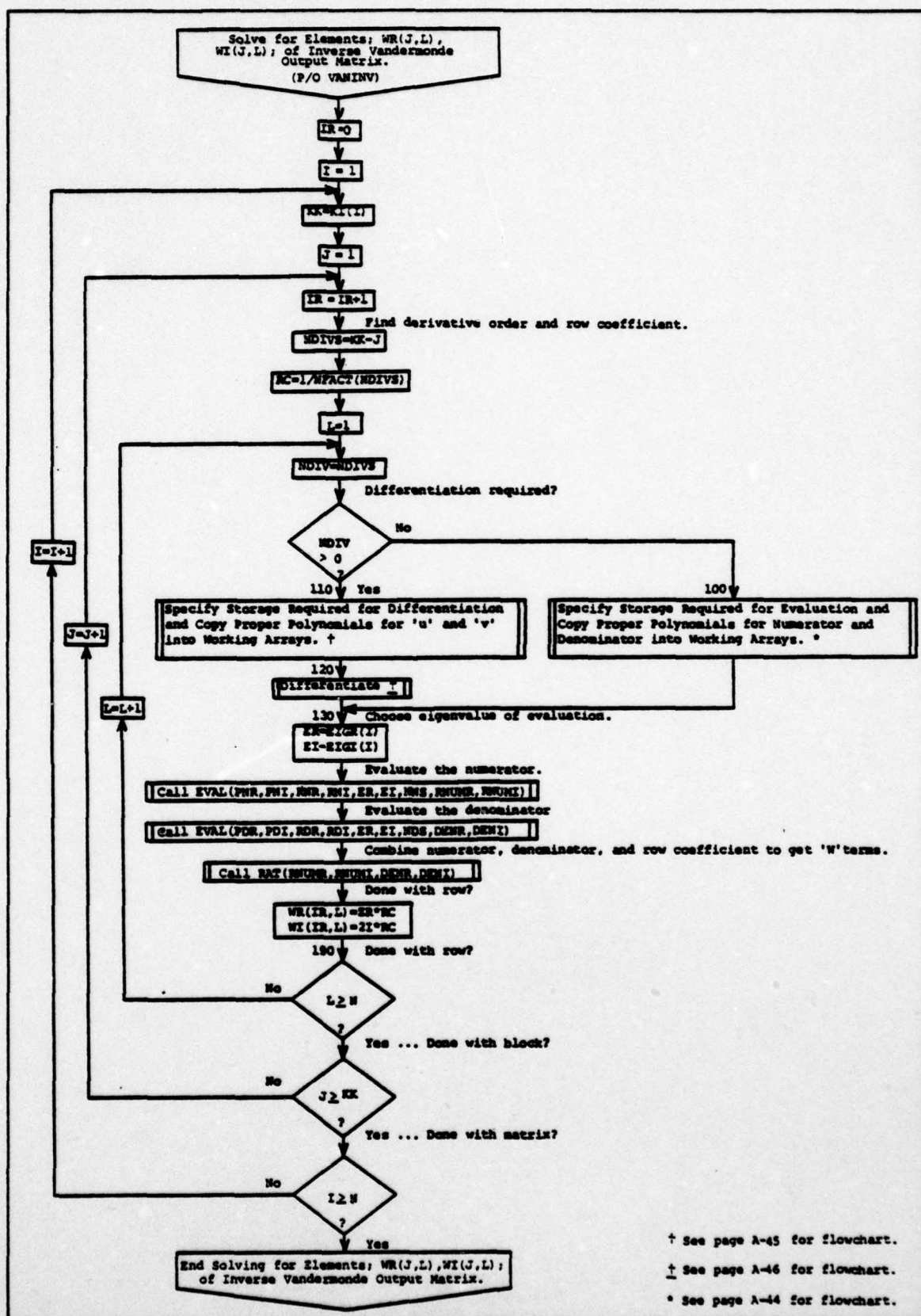


Figure A-21. Flowchart for matrix element computation steps for subroutine VANINV.

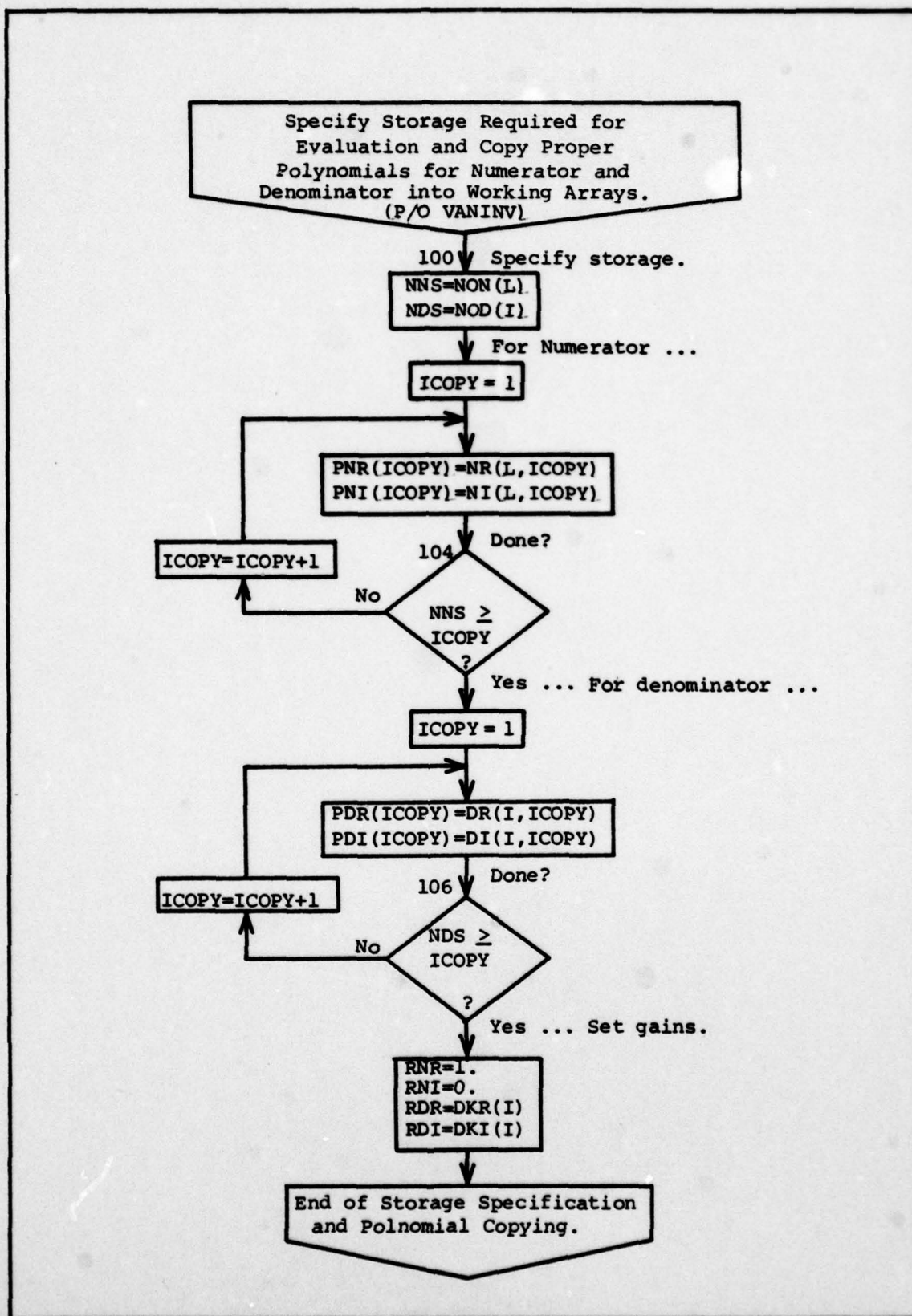


Figure A-22. Flowchart of polynomial transfer steps for subroutine VANINV (if no differentiation required).

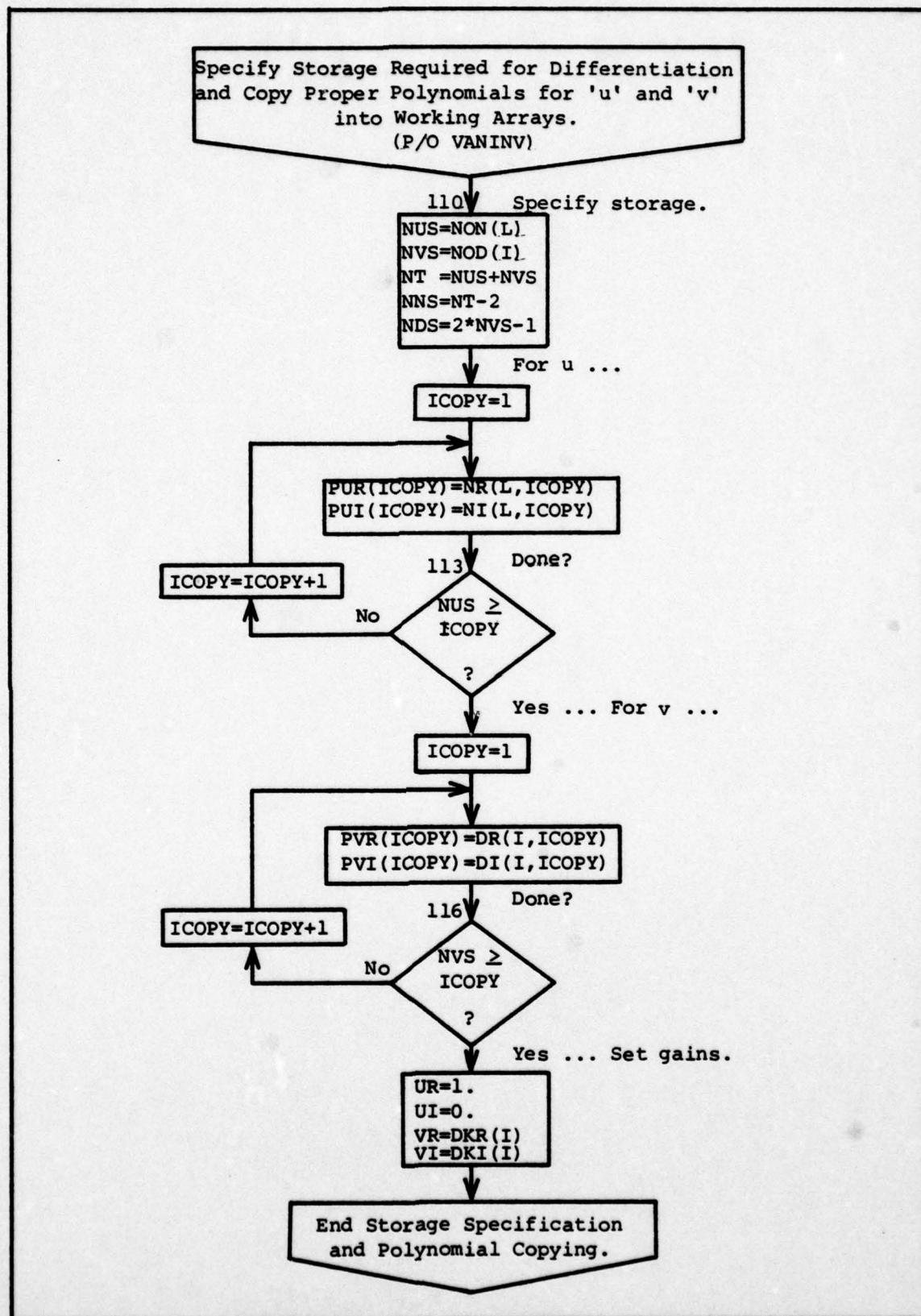


Figure A-23. Flowchart of polynomial transfer steps for subroutine VANINV (if differentiation required).

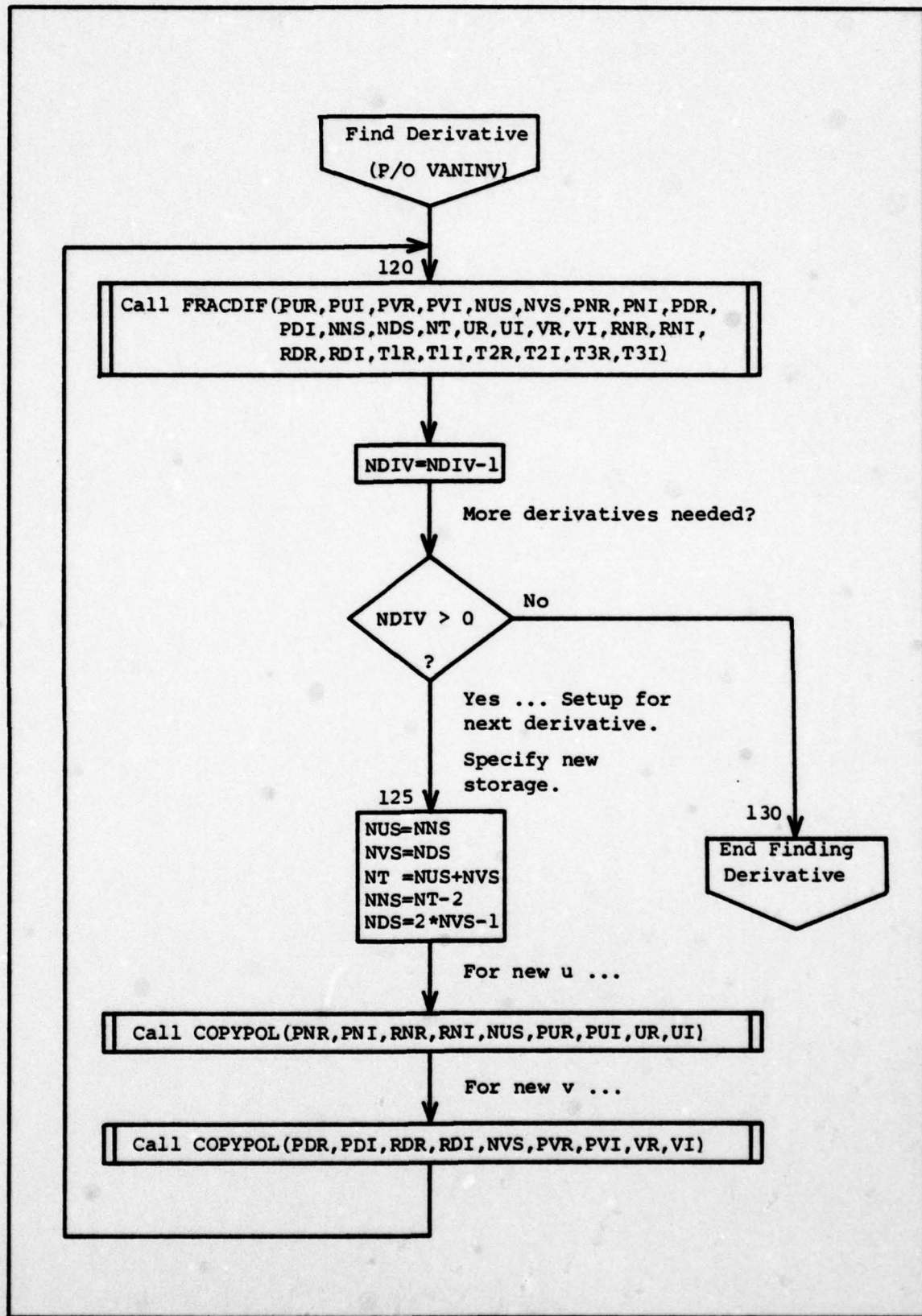


Figure A-24. Flowchart of differentiation steps for subroutine VANINV.

APPENDIX B

CODED PROGRAMS

Preface for Appendix B

Two types of programs are contained in this appendix. Part I contains all the subroutines described in Appendix A. They are listed in alphabetical order and form a complete package of subroutines for implementing Csáki's generalized algorithm as described in Section III of the main text. Part II contains programs and subroutines needed to dimension the arrays for the subroutines of Part I. They also perform input and output operations, and the tests that are described in Section IV of the main text. Since the programs in Part II are not flowcharted, a word description of each is included at the beginning of Part II.

Part I

Contents

| | |
|------------------------------|------|
| Subroutine CGAIN | B-4 |
| Subroutine COPYPOL | B-5 |
| Subroutine ESORT | B-6 |
| Subroutine EVAL | B-8 |
| Subroutine EXPANDC | B-10 |
| Subroutine FRACDIF | B-12 |
| Function NFACT | B-13 |
| Subroutine OUT2C | B-14 |
| Subroutine POLYADD | B-15 |
| Subroutine POLYDIF | B-16 |
| Subroutine POLYMC | B-17 |
| Subroutine POLYSUB | B-19 |
| Subroutine RAT | B-20 |
| Subroutine ROOTAY | B-21 |
| Subroutine UNITY | B-22 |
| Subroutine VANINV | B-24 |


```

C+++++ SUBROUTINE EVAL EVAL
C+++++ SUBROUTINE EVAL EVAL
C+++++ SUBROUTINE EVAL(AI,AKR,AKI,EIGR,EIGI,NS,BR,BI) EVAL
C+++++ DIMENSION AR(NS),AI(NS) EVAL
C+++++ FIND POLYNOMIAL ORDER EVAL
C+++++ N=NS-1 EVAL
C+++++ PUT CONSTANT TERM IN OUTPUT STORAGE EVAL
C+++++ BR=AF(NS) EVAL
C+++++ BI=AI(NS) EVAL
C+++++ CHECK FOR MORE TERMS EVAL
C+++++ IF(N)50,50,10 EVAL
C+++++ 10 CONTINUE EVAL
C+++++ MORE TERMS FOUND...CONTINUE EVALUATION EVAL
C+++++ ER=EIGR EVAL
C+++++ L=N EVAL
C+++++ COMPLEX EIGENVALUE? EVAL
C+++++ IF(EIGI)30,20,30 EVAL
C+++++ 20 CONTINUE EVAL
C+++++ REAL EVALUATION... EVAL
C+++++ DO 25 J=1,L EVAL
C+++++   ADD NEXT HIGHER ORDER TERMS EVAL
C+++++   BR=BR+AR(N)*ER EVAL
C+++++   BI=BI+AI(N)*ER EVAL
C+++++   N=N-1 EVAL
C+++++   UPDATE EVALUATION TERM EVAL
C+++++   25 ER=EF+EIGR EVAL
C+++++   GO TO 50 EVAL
C+++++ 30 CONTINUE EVAL
C+++++ COMPLEX EVALUATION... EVAL
C+++++ EI=EIGI EVAL
C+++++ DO 35 J=1,L EVAL
C+++++   AR=AR(N) EVAL
C+++++   AI=AI(N) EVAL
C+++++   ADD NEXT HIGHER ORDER TERMS EVAL
C+++++   BR=BR+AR(N)*ER-AI(N)*EI EVAL
C+++++   BI=BI+AR(N)*EI+AI(N)*ER EVAL

```

EVAL
EVAL
EVAL
EVAL
EVAL
EVAL
EVAL
EVAL
EVAL
EVAL
EVAL

N=N-1
UPDATE EVALUATION TERMS
TR=EK*EIGR-EI*EIGI
TI=EF+EIGI+EI*EIGR
ER=TF
EI=TI
35 CONTINUE
FO
MULTIPLY BY COMPLEX GAIN
CALL CGAIN(BR,BI,1,AKP,AKI)
RETURN
END

C

C

OUT2C
OUT2C
OUT2C
OUT2C
OUT2C
OUT2C
OUT2C
OUT2C
OUT2C
OUT2C

[illegible]

THIS PAGE IS BEST QUALITY PRINT
FROM COPY FURNISHED TO DDC

POLYMC
POLYMC
POLYMC
POLYMC
POLYMC
POLYMC
POLYMC

CR(IT)=CR(IT)+ARI*BRJ-AII*BIJ
CI(II)=CI(IT)+ARI*BIJ+AII*BRJ
UPDATE GAIN FOR POLYC
CKR=AKR+BKR-AKI*BKI
CKI=AKR+BKI-AKI*BKR
PETUFN
END

90

C


```

C+++++
C SUBROUTINE ROOTAY
C+++++
SUBROUTINE ROOTAY(EIGR,EIGI,K,M,IS,IB,ROOTR,ROOTI)
DIMENSION EIGR(M),EIGI(M),K(M),ROOTR(IS),ROOTI(IS)
IR=1
DO 1 IE=IB,M
L=K(IE)
DO 1 J=1,L
ROOTR(IR)=EIGR(IE)
ROOTI(IR)=EIGI(IE)
IR=IR+1
1 CONTINUE
RETURN
END
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY
ROOTAY

```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC


```

C+++++ C SUBROUTINE UNITY
C+++++
C+++++ SUBROUTINE UNITY(AR,AI,NS,AKR,AKI)
C+++++ COMMON CLOSE,MDIS
C+++++ DIMENSION AR(NS),AI(NS)
C+++++ C CHECK FOR ZERO FIRST COEFFICIENT
C+++++ 10 TAR=AR(1)
C+++++ TAI=AI(1)
C+++++ IF(CLOSE-AR$(TAR))60,60,20
C+++++ MIGHT BE ZERO..CHECK IMAGINARY PART
C+++++ 20 IF(CLOSE-ABS(TAI))70,70,30
C+++++ FIRST COEFFICIENT IS ZERO. DELETE IT AND UPDATE STORAGE SPEC.
C+++++ 30 CONTINUE
C+++++ DO 35 I=1,NS
C+++++ J=I+1
C+++++ AR(I)=AR(J)
C+++++ 35 AI(I)=AI(J)
C+++++ NS=NS-1
C+++++ ARE THERE ANY COEFFICIENTS LEFT?
C+++++ IF(NS)40,40,10
C+++++ NO COEFFICIENTS LEFT...SET GAIN AND CONSTANT TERM TO ZERO, PRINT
C+++++ WARNING & RETURN
C+++++ 40 AKR=0.
C+++++ AKI=0.
C+++++ AR(1)=0.
C+++++ AI(1)=0.
C+++++ MESSAGE PRINTER DISABLED?
C+++++ IF(MDIS)45,45,999
C+++++ 45 PRINT*,**,**
C+++++ PRINT*,** ***WARNING*** SUBROUTINE UNITY WAS GIVEN A NULL POLYNOM",
C+++++ "IAL UPON WHICH TO OPERATE. THE GAIN AND CONSTANT TERM WERE SET ",
C+++++ "TO ZERO AND CONTROL WAS PASSED BACK TO THE CALLING ROUTINE."
C+++++ PRINT*,**,**
C+++++ GO TO 999
C+++++ FIRST COEFFICIENT NOT ZERO...IS IT COMPLEX?
C+++++ 60 IF(CLOSE-ABS(TAI))70,70,62
C+++++ 60 FIRST COEFFICIENT NOT COMPLEX...IS IT UNITY?

```



```

C*****
C SUBROUTINE VANINV
C*****
SUBROUTINE VANINV(EIGR,EIGI,KI,M,NS,MS,WR,WI,NW1,NW2,AR,AI,FR,
+FI,NK,NI,DR,DI,GR,GI,NOD,NON,PNR,PNI,POR,POI,PUR,PVI,T1R,
+T1I,T2R,T2I,T3R,T3I,DKF,DKI)
REAL NR(N,N),NI(N,N)
DIMENSION EIGR(M),EIGI(M),KI(M),AR(N),AI(N),FR(NS),FI(NS),
+DR(M,MS),DI(M,MS),GR(NS),GI(NS),NOD(M),NON(N),PNR(NW1),PNI(NW1),
+POR(NW1),POI(NW1),PUR(NW1),PVI(NW1),T1R(NW1),T1I(NW1),
+T2R(NW2),T2I(NW2),T3R(NW2),T3I(NW2),WR(N,N),WI(N,N),
+DKR(N),DKI(N)

C DATA GNLR,GNLI/1.,0./
C CHECK FOR TRIVIAL SOLUTION (ONLY ONE EIGENVALUE)
C
IF(M-1)10,10,20
10 WR(1,1)=1.
WI(1,1)=0.
PRINT*,"**WARNING** SUBROUTINE VANINV FOUND A TRIVIAL SYSTEM ",
+"WITH ONLY ONE EIGENVALUE: EIGR=",EIGR(1)," EIGI=",EIGI(1),
+" CHECK SYSTEM VALIDITY AND/OR DIMENSION STATEMENTS IN CALLING ",
+"ROUTINE"
GO TO 999
C SET UP DENOMINATOR POLYNOMIALS, D(I)
20 DO 80 I=1,M
C SET UP ROOT ARRAY
IB=2
IT=N-KI(1)
CALL ROOTAY(EIGR,EIGI,KI,M,IT,IA,AR,AI)
C FIND D(I) POLYNOMIAL FROM ROOT ARRAY
IS=IT+1
CALL EXPANDC(AR,AI,IS,FR,FI,GR,GI,GKR,GKI)
C STORE RETURNED POLYNOMIAL AS D(I)
DO 40 K=1,IS
DR(I,K)=GR(K)
DI(I,K)=GI(K)
DKR(I)=GKR

```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC


```

NDIVS=KK-J
RC=1./NFACT(NDIVS)
DO 190 L=1,N
NDIV=NDIVS
C DIFFERENTIATION REQUIRED?
IF(NDIV)100,100,110
100 CONTINUE
C NO...SPECIFY STORAGE REQUIRED FOR EVALUATION AND COPY PROPER
C POLYNOMIALS FOR NUMERATOR & DENOMINATOR INTO WORKING ARRAYS
NNS=NON(L)
NDS=NOD(I)
C FOR NUMERATOR
DO 104 ICOPY=1,NNS
PNR(ICOPY)=NR(L,ICOPY)
104 PNI(ICOPY)=NI(L,ICOPY)
C FOR DENOMINATOR
DO 106 ICOPY=1,NDS
PDR(ICOPY)=DR(I,ICOPY)
106 PDI(ICOPY)=DI(I,ICOPY)
C SET GAINS
RNR=1.
RNI=0.
RDR=DKR(I)
RDI=DKI(I)
GO TO 130
110 CONTINUE
C YES...SPECIFY STORAGE REQUIRED FOR DIFFERENTIATION AND COPY PROPER
C POLYNOMIALS FOR 'U' & 'V' INTO WORKING ARRAYS
NUS=NON(L)
NVS=NOD(I)
NT=NUS+NVS
NNS=NT-2
NDS=2*NVS-1
C FOR U...
DO 113 ICOPY=1,NUS
PUK(ICOPY)=NR(L,ICOPY)
113 PUI(ICOPY)=NI(L,ICOPY)
C FOR V...
DO 116 ICOPY=1,NVS

```

THIS PAGE IS BEST QUALITY PRINT-LEGIBLE
FROM COPY FURNISHED TO DDC


```

PVR(ICOPY)=OR(I,ICOPY)
116 PVI(ICOPY)=DI(I,ICOPY)
C SET GAINS
UR=1.
UI=0.
VR=DKR(I)
VI=DKI(I)
120 CONTINUE
C FIND DERIVATIVE
CALL FRACDIF(PUR,PUI,PVR,PVI,NUS,NVS,PNR,PNI,POR,POI,NNS,NDS,NT,
+UR,UI,VR,VI,RNR,RNI,RDR,RDI,T1R,T1I,T2R,T2I,T3R,T3I)
NDIV=NDIV-1
MORE DERIVATIVES NEEDED?
IF(NDIV)130,130,125
125 CONTINUE
C YES...SET UP FOR NEXT DERIVATIVE
C SPECIFY NEW STORAGE
NUS=NNS
NVS=NDS
NT=NUS+NVS
NNS=NT-2
NDS=2+NVS-1
FOR NEW U...
CALL COPYPOL(PNR,PNI,RNR,RNI,NUS,PUR,PUI,JR,UI)
FOR NEW V...
CALL COPYPOL(PDR,POI,RDR,RDI,NVS,PVR,PVI,VR,VI)
GO TO 120
130 CONTINUE
C NO...CHOOSE EIGENVALUE OF EVALUATION
ER=EIGR(I)
EI=EIGI(I)
C EVALUATE RESULTING NUMERATOR
CALL EVAL(PNR,PNI,RNR,PNI,ER,EI,NNS,RNUMK,RNUMI)
C EVALUATE RESULTING DENOMINATOR
CALL EVAL(PDR,POI,RDR,POI,ER,EI,NDS,DENR,DENI)
C COMBINE NUMERATOR, DENOMINATOR, & ROW COEFFICIENT TO GET ROW TERMS
CALL RAT(RNUMR,RNUMI,DENR,DENI,7R,7I)

```

THIS PAGE IS BEST QUALITY PRINTABLE
FROM COPY FURNISHED TO DDC

VANINV
VANINV
VANINV
VANINV
VANINV

WR(IF,L)=ZR*RC
WI(IF,L)=ZI*RC
100 CONTINUE
999 RETURN
END

Part II

Contents

| | |
|--|------|
| Descriptions of Programs and Subroutines | B-30 |
| Program READER | B-31 |
| Subroutine CATEST | B-32 |
| Subroutine OUT2 | B-35 |
| Subroutine UERTST | B-36 |
| Subroutine VANF | B-37 |

Program READER contains the dimension statements needed to test subroutine VANINV's computational time and accuracy. After reading in the system eigenvalues, READER calls subroutine CATEST to perform the tests. In the form shown, READER works only with real, distinct eigenvalues. Some simple modifications would enable it to accommodate complex and/or repeated eigenvalues.

Subroutine CATEST performs computation accuracy and time tests on subroutine VANINV and the IMSL subroutine LINV2F.

Subroutine OUT2 prints out, in column form, a two dimensional array. It is similar to OUT2C shown in Part I.

Subroutine UERTST disables the warning messages from the IMSL routines. Terminal errors (IER > 128) are not suppressed.

Subroutine VANF forms the Vandermonde matrix from a collection of eigenvalues. In its present form, it cannot handle complex or repeated eigenvalues.


```

PROGAM READER(INPUT=/72, OUTPUT, TAPE5=INPUT, TAPE6=OUTPUT)
COMMON CLOSE, MDIS
DIMENSION EIGR(20), EIGI(20), KI(20), AR(20), AI(20), FR(21), FI(21),
+DR(20,21), DI(20,21), GR(21), GI(21), NOD(20), NON(20), PNR(20), PNI(20),
+POR(20), POI(20), PUR(20), PUI(20), PVR(20), PVI(20), T1R(40), T1I(40),
+T2R(40), T2I(40), T3R(40), T3I(40), VANIR(20,20), VANII(20,20), OKR(20),
+DKI(20), C(20,20), B(20,20), VANR(20,20), NR(20,20), NI(20,20),
+WKAREA(500)
DATA ID,IDI/4HVNAR,5HCHECK/
CLOSE=.0000001
MDIS=1
READ*,NO,NUMB,IDGT
PRINT*, "THERE ARE ",NO," DATA SETS. EACH WILL BE TESTED FOR ",
+"COMPUTATION TIME BY PERFORMING ",NUMB," ITERATIONS WITH IDGT=",
+IDGT
PRINT*, "*****"
DO 50 I=1,NO
READ*,N
PRINT*, "DATA SET ",I," HAS ",N," EIGENVALUES. THEY ARE:"
DO 10 J=1,N
READ*,EIGR(J)
EIGI(J)=0.
KI(J)=1
10 PRINT*, "EIGR(",J,") = ",EIGR(J), " EIGI(",J,") = ",EIGI(J), " KI(",J,
+",") = ",KI(J)
PRINT*, "*****"
NS=N+1
NW2=N+N
CALL CATEST(EIGR,EIGI,KI,VANIR,VANII,AR,AI,FR,FI,NR,NI,DR,DI,GR,
+GI,NCD,NON,PNR,PNI,PDR,POI,PUR,PUI,PVR,PVI,T1R,T1I,T2R,T2I,T3R,
+T3I,OKR,DKI,NS,N,C,NW2,B,ID,IDI,NUMB,VANR,IDGT,WKAREA)
20 PRINT*, "*****"
PRINT*, "END"
END

```

```

C+++++ CATEST
C SUBROUTINE CATEST
C+++++
SUBROUTINE CATEST(EIGR,EIGI,KI,VANIR,VANII,AR,AI,FR,FI,NI,DR,
+DI,GF,GI,NOD,NON,PNR,PNI,POR,PDI,PUR,PUI,PVR,PVI,T1I,T2R,T2I,
+T3K,T3I,DKR,DKI,NS,N,C,NW2,B,ID,IDI,NUMB,VANR,IDGT,WKAREA)
DIMENSION EIGR(N),EIGI(N),KI(N),AR(N),AI(N),FR(NS),FI(NS),
+DR(N,NS),DI(N,NS),GR(NS),GI(NS),NOD(N),NON(N),PNR(N),PNI(N),
+PDR(N),PDI(N),PUR(N),PUI(N),PVR(N),PVI(N),T1R(NW2),T1I(NW2),
+T2R(NW2),T2I(NW2),T3R(NW2),T3I(NW2),VANIR(N,N),VANII(N,N),DKR(N),
+DKI(N),C(N,N),B(N,N),VANR(N,N)
IDGTS=IDGT
CALL VANF(EIGR,N,VANR)
CALL OUT2(N,N,VANR,ID)
CPSTORE=SECOND(CP)
DO 10 I=1,NUMB
10 CALL VANINV(EIGR,EIGI,KI,N,N,NS,NS,VANIR,VANII,N,NW2,AR,AI,FR,FI,
+NR,NI,DR,DI,GR,GI,NOD,NON,PNR,PNI,POR,PDI,PUR,PUI,PVR,PVI,T1R,T1I,
+T2R,T2I,T3R,T3I,DKR,DKI)
CPTIME=SECOND(CP)-CPSTORE
PRINT*,"4**"
PRINT*,"TIME FOR ",NUMB," ITERATIONS USING VANINV IS: ",CPTIME
PRINT*,"**"
PRINT*,"INVERSE VANDERMONDE FOUND IS:"
CALL OUT2(N,N,VANIR,ID)
PRINT*,"**"
CALL VMULFF(VANR,VANIR,N,N,N,N,N,C,N,IER)
PRINT*,"FOR VMULFF, IEP= ",IER
DO 20 I=1,N
20 C(I,1)=C(I,I)-1.
DO 25 J=1,N
DO 25 I=1,N
C(I,J)=ABS(C(I,J))
25 VANIR(I,J)=ABS(VANIR(I,J))
CALL VMULFF(VANIR,C,N,N,N,N,N,VANII,N,IEF)
PRINT*,"FOR VMULFF(2), IER= ",IER
PRINT*,"ACCURACY CHECK MATRIX (CHECK=A*(-1)**E) SHOULD BE NULL."

```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST
CATEST

```

DO 55 I=1,N
  C(I,J)=ABS(C(I,J))
  B(I,J)=ABS(B(I,J))
  CALL VMULFF(B,C,N,N,N,N,VANII,N,IER)
  PRINT*, "FOR VMULFF(2), IER=", IER
  PRINT*, "ACCURACY CHECK MATRIX (CHECK=A*(-1)*E) SHOULD BE NULL."
  CALL OUT2(N,N,VANII,IDI)
  PRINT*, "*****"
  SUM=0.
DO 60 J=1,N
DO 60 I=1,N
  SUM=SUM+VANII(I,J)
PRINT*, "ACCURACY MERIT FIGURE FOR INSL IS: ", SUM
PRINT*, "*****"
RETURN
END

```

THIS PAGE IS BEST QUALITY PRACTITIONER
FROM COPY FURNISHED TO DDC

```

C+++++*****
C SUBROUTINE OUT2
C+++++*****
SUBROUTINE OUT2(NR,NC,A,ID)
DIMENSION A(NR,NC)
DO 10 J=1,NC
DO 10 I=1,NR
WRITE 5,ID,I,J,A(I,J)
5 FORMAT(1X,A5,"(",I2,"",I2,"")= ",E25.18)
10 CONTINUE
RETURN
END
OUT2
OUT2
OUT2
OUT2
OUT2
OUT2
OUT2
OUT2
OUT2
OUT2
OUT2
OUT2

```

UERTST
UERTST
UERTST
UERTST
UERTST
UERTST
UERTST
UERTST

```
C+++++  
C SUBROUTINE UERTST  
C+++++  
C+++++  
SUBROUTINE UERTST (IER, NAME)  
IF (IER.LT.128) RETURN  
PRINT*, "UERTST TERMINAL ERROR FROM--", NAME, " IER= ", IER  
RETURN  
END
```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO HQC


```

C+++++SUBROUTINE VANF
C SUBROUTINE VANF
C+++++
C SUBROUTINE VANF(EIGR,N,VANP)
  DIMENSION EIGR(N),VANR(N,N)
  PUT 1'S IN FIRST ROW
  DO 10 J=1,N
    10 VANR(1,J)=1.
  C FILL IN REMAINDER OF EACH COLUMN
  DO 30 J=1,N
    FI=EIGR(J)
    TR=1.
    DO 30 I=2,N
      TR=TR*FI
    30 VANR(I,J)=TR
  RETURN
  END
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF
VANF

```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

APPENDIX C

SUPPLEMENTARY MATHEMATICAL DEVELOPMENTS

Three mathematical developments are included here to aid in understanding some concepts which were stated in the text. They are placed here in order to avoid long interruptions in the main text. The concepts are:

| | |
|---|-----|
| C1. Sensitivity Derivatives as Part of the Augmented System State Transition Matrix | C-3 |
| C2. Determination of State Transition Matrices Using the Cayley-Hamilton Theorem | C-4 |
| C3. Definition and Construction of Component Matrices | C-9 |

C1. Sensitivity Derivatives as Part of the Augmented System State Transition Matrix

Using Eq(8), the system plant matrix may be augmented to find the sensitivities of the i'th parameter as

$$\bar{A}_i \equiv \begin{bmatrix} A & | & 0 \\ \hline A_{(i)} & | & A \end{bmatrix} \quad 2n \times 2n \quad (C1-1)$$

Notice that the eigenvalues, and hence the system modes, have not changed. The augmented system has n pseudo modes in addition to the original n modes, but they are entirely independent of the original modes in the component matrix approach, and the augmentation therefore does not actually change the system. This desirable feature of the component matrix approach is illustrated in Appendix D, Example 3.

From Eq (C1-1) the state transition matrix for the augmented system can be defined. For time-invariant systems, the exponential can be applied to each section of the partitioned matrix. Thus

$$e^{\bar{A}_i t} = \begin{bmatrix} e^{At} & | & 0 \\ \hline e^{A_{(i)}t} & | & e^{At} \end{bmatrix} \quad 2n \times 2n \quad (C1-2)$$

which shows that the sensitivity derivative of the original state transition matrix is one of the partitioned sections of the state transition matrix for the augmented system.

C2. Determination of State Transition Matrices Using the Cayley-Hamilton Theorem

This section shows how the Cayley-Hamilton theorem can be applied in order to determine a state transition matrix (STM). The use of the Vandermonde matrix and remainder polynomial coefficients is described for this application. Finally, the method is extended to show how it can be used to find the partitioned STM for system sensitivity calculations. Numerical examples of the concepts presented here are included in Appendix D, Examples 1 and 2.

Basic Cayley-Hamilton Theorem Application. The Cayley-Hamilton theorem states that any square matrix satisfies its own characteristic equation. From it, the STM can be shown to be a summation of products of the "remainder polynomial" coefficients, $a_j(t)$, and powers of the system matrix, A , (Ref 9:V-110 through V-115b):

$$e^{At} = \sum_{j=1}^n A^{j-1} a_j(t) \quad (C2-1)$$

The remainder polynomial coefficients can be uniquely determined from n linear "modal" equations (Ref 9:V-115b):

$$\begin{bmatrix} e^{\lambda_1 t} \\ e^{\lambda_2 t} \\ \vdots \\ e^{\lambda_n t} \end{bmatrix} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_n & \lambda_n^2 & \cdots & \lambda_n^{n-1} \end{bmatrix} \begin{bmatrix} a_1(t) \\ a_2(t) \\ \vdots \\ a_n(t) \end{bmatrix} \quad (C2-2)$$

where $e^{\lambda_j t}$ are system "modes" and λ_j are system eigenvalues. Eq (C2-2) is equivalent to

$$\underline{e}^{\lambda_j t} = \underline{v}^T \underline{a}_j(t) \quad (C2-3)$$

where \underline{v} is the "Vandermonde" matrix. Since \underline{v}^T must be inverted to find the remainder polynomial coefficients, it must not be singular. Therefore, if the system matrix does not have n distinct eigenvalues (ie: if at least one eigenvalue, λ_j , has a multiplicity, m_j , greater than one), \underline{v}^T is not directly invertible if defined as shown in Eq (C2-2). Thus for systems having repeated eigenvalues, the Vandermonde matrix is defined by taking repeated derivatives, w.r.t. the eigenvalue, of the modal equation for that eigenvalue until there is one independent equation for each occurrence of the eigenvalue (Ref 9:V-116). For example, in a system having an eigenvalue with multiplicity $m_j=3$, the original modal equation must be differentiated twice to get three independent equations:

$$\frac{d}{d\lambda_j} e^{\lambda_j t} = t e^{\lambda_j t} = \frac{d}{d\lambda_j} \begin{bmatrix} 1 & \lambda_j & \lambda_j^2 & \dots & \lambda_j^{n-1} \end{bmatrix} \begin{bmatrix} a_1(t) \\ a_2(t) \\ \vdots \\ a_n(t) \end{bmatrix} \quad (C2-4)$$

and

$$\frac{d^2}{d\lambda_j^2} e^{\lambda_j t} = t^2 e^{\lambda_j t} = \frac{d^2}{d\lambda_j^2} \begin{bmatrix} 1 & \lambda_j & \lambda_j^2 & \dots & \lambda_j^{n-1} \end{bmatrix} \begin{bmatrix} a_1(t) \\ a_2(t) \\ \cdot \\ \cdot \\ a_n(t) \end{bmatrix} \quad (C2-5)$$

Assuming λ_1 is the repeated eigenvalue in the example, Eq (C2-2) is then

$$\begin{bmatrix} e^{\lambda_1 t} \\ t e^{\lambda_1 t} \\ t^2 e^{\lambda_1 t} \\ e^{\lambda_1 t} \\ \cdot \\ e^{\lambda_n t} \end{bmatrix} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 0 & 1 & 2\lambda_1 & \dots & (n-1)\lambda_1^{n-2} \\ 0 & 0 & 2 & \dots & (n-2)(n-1)\lambda_1^{n-3} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \lambda_n & \lambda_n^2 & \dots & \lambda_n^{n-1} \end{bmatrix} \begin{bmatrix} a_1(t) \\ a_2(t) \\ a_3(t) \\ a_4(t) \\ \cdot \\ a_n(t) \end{bmatrix} \quad (C2-6)$$

For simplicity, only systems with distinct eigenvalues are used in the remainder of this discussion.

By defining v_{ij} to be the element in the i 'th row and j 'th column of the inverse of the transposed Vandermonde matrix, the remainder polynomial coefficients may be determined by

$$\begin{bmatrix} a_1(t) \\ a_2(t) \\ . \\ . \\ . \\ a_n(t) \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ v_{n1} & v_{n2} & \cdots & v_{nn} \end{bmatrix} \begin{bmatrix} e^{\lambda_1 t} \\ e^{\lambda_2 t} \\ . \\ . \\ . \\ e^{\lambda_n t} \end{bmatrix} \quad (C2-7)$$

Eq (C2-1) may be expanded to

$$e^{At} = a_1(t)I + a_2(t)A + a_3(t)A^2 + \dots + a_n(t)A^{n-1} \quad (C2-8)$$

Applying the results of Eq (C2-7) to Eq (C2-8) results in

$$\begin{aligned}
 e^{At} = & (v_{11} e^{\lambda_1 t} I + v_{12} e^{\lambda_2 t} I + \dots + v_{1n} e^{\lambda_n t} I) \\
 & + (v_{21} e^{\lambda_1 t} A + v_{22} e^{\lambda_2 t} A + \dots + v_{2n} e^{\lambda_n t} A) \\
 & + \dots \\
 & + (v_{n1} e^{\lambda_1 t} A^{n-1} + v_{n2} e^{\lambda_2 t} A^{n-1} + \dots + v_{nn} e^{\lambda_n t} A^{n-1})
 \end{aligned} \quad (C2-9)$$

Notice that in Eq (C2-9) the state transition matrix is expressed in terms of products of: (1) elements of the inverse transposed Vandermonde matrix, (2) the system modes, and (3) powers of the original system matrix. The remainder polynomial coefficients, though they were used in the theoretical formulation of the solution, do not even appear in it and thus it is never necessary to actually calculate them. A very significant result of this formulation is that the

original system modes are completely separate and visible, thus physical insight into the system is preserved. This decoupling of modes makes it possible to represent the state transition matrix in terms of the modes, each with a "component matrix" multiplier. An explanation of component matrix formation is given in section C3.

Extended Application of the Cayley-Hamilton Theorem. The procedure outlined in the preceding discussion can be very simply extended to the augmented system state transition matrix, Eq (16), which is used in the sensitivity parameter calculations for the system identification process. Eq (16) is repeated here for the reader's benefit:

$$e^{\begin{bmatrix} A & | & 0 \\ \hline A_{(1)} & | & A \end{bmatrix} t} = \sum_{j=1}^{2n} \begin{bmatrix} A & | & 0 \\ \hline A_{(1)} & | & A \end{bmatrix} a_j(t) \quad (C2-10)$$

Note from Eq (C2-10) that the augmented system has the same eigenvalues as the original system since the augmented system matrix is triangular. The only difference is that the eigenvalue multiplicities of the augmented system are double those of the original system. This means that the Vandermonde matrix for the augmented system is $2n \times 2n$, there are $2n$ remainder polynomial coefficients, and there are $2n$ modes. However, half of the modes are simply derivative modes resulting from the doubled eigenvalue multiplicity which came from augmenting the system. These extra modes ultimately have null component matrices (Ref 9:V-122) and may be thought of as placeholders or "pseudo" modes.

Thus it is seen how the Cayley-Hamilton theorem can be applied to determine the state transition matrices of both the original and augmented (sensitivity) systems. The next step is to determine how to

find the "component matrices" which were revealed as part of Cayley-Hamilton theorem application. That determination is the subject of the next section.

C3. Definition and Construction of Component Matrices. Several references have been made to "component matrices" in other sections of this thesis. The purpose of this section is to show what component matrices are and how they are formed.

A component matrix may be defined as a matrix which shows the contribution of a given system mode to the state transition matrix (Ref 15:610). Thus it can be seen from Eq (C2-9) that each column of terms constitutes a component matrix, multiplied by its corresponding mode.

Component matrices are commonly labeled with a double subscript, e.g.: Z_{10} . The first subscript identifies which eigenvalue made the mode with which the component matrix belongs. The second subscript can be nonzero only when the system has repeated eigenvalues. It then identifies the number of derivatives which must be taken of the associated primary mode to obtain the pseudo mode with which the component matrix belongs. (This derivative process is described in Section C2.) Two examples should help clarify this. First consider Z_{10} . This is the component matrix associated with the primary mode of eigenvalue number one: $e^{\lambda_1 t}$. Next consider Z_{82} . This is the component matrix associated with one of the pseudo modes of eigenvalue number eight. Specifically, it is for the pseudo mode found by taking two time derivatives of the primary mode: $\frac{d^2}{dt^2} e^{\lambda_8 t} = t^2 e^{\lambda_8 t}$. Note that this implies eigenvalue eight has a multiplicity of at least three.

Applying the observations made and the subscript conventions just described results in a component matrix formula:

$$z_{j,k} = \sum_{\ell=1}^n v_{\ell,c} A^{\ell-1}; \quad c = \begin{bmatrix} j \\ \sum_{i=1}^j m_i \end{bmatrix} - m_j + k + 1 \quad (C3-1)$$

where $0 \leq k \leq m_j - 1$; m_j being the multiplicity of the j 'th eigenvalue.

The state transition matrix may then be found from

$$e^{At} = \sum_{j=1}^M \sum_{k=0}^{m_j-1} (z_{j,k}) t^k e^{\lambda_j t} \quad (C3-2)$$

in which M is the number of different eigenvalues. Some numerical examples of the formation of component matrices are given in Appendix D, Examples 1 and 2.

APPENDIX D

EXAMPLE PROBLEMS

Several numerical examples are provided here to help clarify the application of concepts presented in the text. Examples 1 and 2 work with systems whose eigenvalues were carefully selected for illustrative purposes. Each starts with a set of eigenvalues, λ_i , and a plant matrix, A , which contains a set of system parameters, θ_i . For the system identification process outlined in the text, the quantities sought are the output sensitivities, $\frac{\partial y}{\partial \theta_i}$, as defined by Eq (13). Example 4 illustrates the use of Csáki's generalized algorithm. This is also the process used by subroutine VANINV.

Example 1 - Distinct Eigenvalue System

Suppose $n = 3$, and

$$\begin{array}{ll} \lambda_1 = 4 & \theta_1 = -1 \\ \lambda_2 = -3 & \theta_2 = -2 \\ \lambda_3 = 1 & \end{array} \quad A = \begin{bmatrix} \lambda_1 & 0 & 0 \\ \theta_1 & \lambda_2 & 0 \\ \theta_2 & 0 & \lambda_3 \end{bmatrix} \quad (D1-1)$$

The solution is presented in four steps:

1. Determination of inverse Vandermonde matrix and $(n-1)$ powers of the plant matrix.
2. Calculation of the state transition matrix, e^{At} .
3. Calculation of the parameter sensitivity derivatives, $e_{(i)}^{At}$, of the state transition matrix.
4. Determination of e^{At} and $e_{(i)}^{At}$ by means of component matrices rather than the solution of n linear equations as done for steps 2 and 3.

Inverse Vandermonde Matrix. For distinct eigenvalues, no derivatives need be taken to find the Vandermonde matrix:

$$V^T = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 \\ 1 & \lambda_2 & \lambda_2^2 \\ 1 & \lambda_3 & \lambda_3^2 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 16 \\ 1 & -3 & 9 \\ 1 & 1 & 1 \end{bmatrix} \quad (D1-2)$$

which has an inverse

$$[V^T]^{-1} = \begin{bmatrix} -1/7 & 1/7 & 1 \\ 2/21 & -5/28 & 1/12 \\ 1/21 & 1/28 & -1/12 \end{bmatrix} \quad (D1-3)$$

Powers of Plant Matrix. Since $n=3$, only powers of A through $n-1 = 2$ need be found. If A^2 is multiplied out, the element in the i 'th row and j 'th column of the product may be represented by

$$[A^2]_{ij} = \sum_{k=1}^{n=3} \alpha_{ik} \alpha_{kj} \quad (D1-4)$$

where the α 's represent the elements of the original matrix. For this example:

$$A^2 = \begin{bmatrix} \lambda_1 & 0 & 0 \\ \theta_1 & \lambda_2 & 0 \\ \theta_2 & 0 & \lambda_3 \end{bmatrix} \begin{bmatrix} \lambda & 0 & 0 \\ \theta_1 & \lambda_2 & 0 \\ \theta_2 & 0 & \lambda_3 \end{bmatrix} = \begin{bmatrix} \lambda_1^2 & 0 & 0 \\ \theta_1(\lambda_1 + \lambda_2) & \lambda_2^2 & 0 \\ \theta_2(\lambda_1 + \lambda_3) & 0 & \lambda_3^2 \end{bmatrix} \quad (D1-5)$$

or

$$A^2 = \begin{bmatrix} 4 & 0 & 0 \\ -1 & -3 & 0 \\ -2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 \\ -1 & -3 & 0 \\ -2 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 16 & 0 & 0 \\ -1 & 9 & 0 \\ -10 & 0 & 1 \end{bmatrix} \quad (D1-6)$$

State Transition Matrix. Eq (C2-8) for this example is

$$e^{At} = a_1(t)I + a_2(t)A + a_3(t)A^2 \quad (D1-7)$$

The remainder polynomial coefficients, $a_j(t)$, may be found by applying Eq (C2-7):

$$\begin{bmatrix} a_1(t) \\ a_2(t) \\ a_3(t) \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{bmatrix} \begin{bmatrix} \lambda_1 t \\ e^{\lambda_1 t} \\ \lambda_2 t \\ e^{\lambda_2 t} \\ \lambda_3 t \\ e^{\lambda_3 t} \end{bmatrix} \quad (D1-8)$$

This results in

$$\begin{aligned} a_1(t) &= -1/7 e^{4t} + 1/7 e^{-3t} + e^t \\ a_2(t) &= 2/21 e^{4t} - 5/28 e^{-3t} + 1/12 e^t \\ a_3(t) &= 1/21 e^{4t} + 1/28 e^{-3t} - 1/12 e^t \end{aligned} \quad (D1-9)$$

Substituting into Eq (D2-7):

$$\begin{aligned} e^{At} &= (-1/7 e^{4t} + 1/7 e^{-3t} + e^t) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &+ (2/21 e^{4t} - 5/28 e^{-3t} + 1/12 e^t) \begin{bmatrix} 4 & 0 & 0 \\ \theta_1 & -3 & 0 \\ \theta_2 & 0 & 1 \end{bmatrix} \\ &+ (1/21 e^{4t} + 1/28 e^{-3t} - 1/12 e^t) \begin{bmatrix} 16 & 0 & 0 \\ \theta_1 & 9 & 0 \\ 5\theta_2 & 0 & 1 \end{bmatrix} \end{aligned} \quad (D1-10)$$

(The parameter sensitivities, θ_i , are left in for the present so the partials of the state transition matrix w.r.t. the parameter sensitivities can be taken later.)

Adding everything up results in

$$e^{At} = \begin{bmatrix} e^{4t} & 0 & 0 \\ (-\frac{\theta_1}{7} e^{4t} - \frac{\theta_1}{7} e^{-3t}) & e^{-3t} & 0 \\ (\frac{\theta_2}{3} e^{4t} - \frac{\theta_2}{3} e^t) & 0 & e^t \end{bmatrix} \quad (D1-11)$$

Which when evaluated at the values of θ_i yields

$$e^{At} = \begin{bmatrix} e^{4t} & 0 & 0 \\ (-1/7e^{4t} + 1/7e^{-3t}) & e^{-3t} & 0 \\ (-2/3e^{4t} + 2/3e^t) & 0 & e^t \end{bmatrix} \quad (D1-12)$$

Parameter Sensitivity Derivatives. The parameter sensitivity derivatives, $e_{(i)}^{At}$, of the state transition matrix can be found by differentiating Eq (D1-11) w.r.t. the parameters, θ_i . For θ_1 :

$$e_{(1)}^{At} = \begin{bmatrix} 0 & 0 & 0 \\ (1/7e^{4t} - 1/7e^{-3t}) & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (D1-13)$$

And for θ_2 :

$$e_{(2)}^{At} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ (1/3e^{4t} - 1/3e^t) & 0 & 0 \end{bmatrix} \quad (D1-14)$$

Use of Component Matrices. There exists one component matrix for each system mode or pseudo mode, therefore it is necessary to find three component matrices for this system. Eq (C3-2), which is repeated here as Eq (D1-15) may be used to find the state transition matrix.

$$e^{At} = \sum_{j=1}^M \sum_{k=0}^{m_j-1} (Z_{j,k}) t^k e^{\lambda_j t} \quad (D1-15)$$

For this example, there are $M=3$ distinct eigenvalues. There are no pseudo modes, therefore $k=0$ always. The three component matrices to be found are Z_{10} , Z_{20} and Z_{30} , which are found by using Eq (C3-1), repeated here as (D1-16):

$$Z_{j,k} = \sum_{\ell=1}^n v_{\ell,c} A^{\ell-1}; \quad c = \left[\begin{matrix} j \\ \ell, m_i \end{matrix} \right]_{i=1}^j - m_j + k + 1 \quad (D1-16)$$

Therefore, substituting from previous calculations;

$$Z_{10} = \sum_{\ell=1}^n v_{\ell,1} A^{\ell-1} \quad (D1-17a)$$

$$= -\frac{1}{7} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \frac{2}{21} \begin{bmatrix} 4 & 0 & 0 \\ \theta_1 & -3 & 0 \\ \theta_2 & 0 & 1 \end{bmatrix}$$

$$+ \frac{1}{21} \begin{bmatrix} 16 & 0 & 0 \\ \theta_1 & 9 & 0 \\ 5\theta_2 & 0 & 1 \end{bmatrix} \quad (D1-17b)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{7}\theta_2 & 0 & 0 \\ \frac{1}{3}\theta_2 & 0 & 0 \end{bmatrix} \quad (D1-17c)$$

$$Z_{20} = \sum_{\ell=1}^{n=3} v_{\ell,2} A^{\ell-1} \quad (D1-18a)$$

$$= \frac{1}{7} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{5}{28} \begin{bmatrix} 4 & 0 & 0 \\ \theta_1 & -3 & 0 \\ \theta_2 & 0 & 1 \end{bmatrix} + \frac{1}{28} \begin{bmatrix} 16 & 0 & 0 \\ \theta_1 & 9 & 0 \\ 5\theta_2 & 0 & 1 \end{bmatrix} \quad (D1-18b)$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ -\frac{1}{7}\theta_1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (D1-18c)$$

$$z_{30} = \sum_{\ell=1}^{n=3} v_{\ell,3} A^{\ell-1} \quad (D1-19a)$$

$$= 1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \frac{1}{12} \begin{bmatrix} 4 & 0 & 0 \\ \theta_1 & -3 & 0 \\ \theta_2 & 0 & 1 \end{bmatrix} - \frac{1}{12} \begin{bmatrix} 16 & 0 & 0 \\ \theta_1 & 9 & 0 \\ 5\theta_2 & 0 & 1 \end{bmatrix} \quad (D1-19b)$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -\frac{1}{3}\theta_2 & 0 & 1 \end{bmatrix} \quad (D1-19c)$$

Substituting the component matrices into Eq (D1-15) results in the state transition matrix:

$$e^{At} = \sum_{j=1}^{M=3} \sum_{k=0}^0 (z_{j,0})^k t^k e^{\lambda_j t} \quad (D1-20a)$$

$$= z_{10} e^{\lambda_1 t} + z_{20} e^{\lambda_2 t} + z_{30} e^{\lambda_3 t} \quad (D1-20b)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{7}\theta_1 & 0 & 0 \\ \frac{1}{3}\theta_2 & 0 & 0 \end{bmatrix} e^{4t} + \begin{bmatrix} 0 & 0 & 0 \\ -\frac{1}{7}\theta_1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} e^{-3t}$$

$$+ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -\frac{1}{3}\theta_2 & 0 & 1 \end{bmatrix} e^t \quad (D1-20c)$$

$$= \begin{bmatrix} (e^{4t} & &) & 0 & 0 \\ (\frac{1}{7}\theta_1 e^{4t} & \frac{1}{7}\theta_1 e^{-3t} &) & e^{-3t} & 0 \\ (\frac{1}{3}\theta_2 e^{4t} & & -\frac{1}{3}\theta_2 e^t) & 0 & e^t \end{bmatrix} \quad (D1-20d)$$

Comparing Eq (D1-20d) with Eq (D1-11) shows that the same answer comes out either way. Note from Eq (D1-20c) how clearly visible the modes are. Their individual influence on the state transition matrix is easily discernable. Also, the influence of the parameters, θ_i , on both the modes and the state transition matrix are easily seen from this formulation.

Note that in actual practice the parameters are numbers rather than variables as shown in this example. They were carried as variables in the example to show how they influence the modes and component matrices. Another important point to note is how direct the component matrix approach turns out to be in the end. Although the development may seem to follow several independent paths, the application is very direct. Once the component matrices are found from the inverse Vandermonde matrix, all the effects of the modes and parameter sensitivities are related to the system by Eq (21).

Example 2 - Augmented System

If $n = 2$, and

$$\begin{aligned} \lambda_1 &= 2 & \theta_1 &= -3 \\ \lambda_2 &= -1 \end{aligned} \quad A = \begin{bmatrix} \lambda_1 & 0 \\ \theta_1^2 & \lambda_2 \end{bmatrix} \quad (D2-1)$$

the solution is found as shown below, following the same procedure as Example 1. Before proceeding, however, the augmented system must be formed

$$\bar{A} = \left[\begin{array}{cc|c} A & & 0 \\ \hline & & \\ \hline A_{(1)} & & A \end{array} \right] = \left[\begin{array}{cc|c|c} \lambda_1 & 0 & & 0 \\ \theta_1^2 & \lambda_2 & & \\ \hline 0 & 0 & \lambda_1 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & \lambda_2 \end{array} \right] \quad (D2-2)$$

Notice that the eigenvalues of the augmented system are the same as those of the original system. The only difference is that the multiplicity of each is doubled.

Inverse Vandermonde Matrix. Since the eigenvalues of the augmented system are not distinct, derivatives must be taken to form the transposed Vandermonde matrix:

$$V^T = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \lambda_1^3 \\ 0 & 1 & 2\lambda_1 & 3\lambda_1^2 \\ 1 & \lambda_2 & \lambda_2^2 & \lambda_2^3 \\ 0 & 1 & 2\lambda_2 & 3\lambda_2^2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 4 & 8 \\ 0 & 1 & 4 & 12 \\ 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 \end{bmatrix} \quad (D2-3)$$

which has an inverse

$$[V^T]^{-1} = \begin{bmatrix} 21/81 & -18/81 & 60/81 & 36/81 \\ 36/81 & -27/81 & -36/81 & 0. \\ 9/81 & 0. & -9/81 & -27/81 \\ -6/81 & 9/81 & 6/81 & 9/81 \end{bmatrix} \quad (D2-4)$$

Powers of Plant Matrix. For this system with $n=4$, powers of A through A^3 must be taken:

$$\bar{A}^{-2} = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ \theta_1^2 & \lambda_2 & 0 & 0 \\ a & 0 & \lambda_1 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & \lambda_2 \end{bmatrix} - \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ \theta_1^2 & \lambda_2 & 0 & 0 \\ a & 0 & \lambda_1 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & \lambda_2 \end{bmatrix} \quad (D2-5a)$$

$$= \begin{bmatrix} \lambda_1^2 & a & 0 & 0 \\ \theta_1^2(\lambda_1 + \lambda_2) & \lambda_2^2 & 0 & 0 \\ a & a & \lambda_1^2 & 0 \\ 2\theta_1(\lambda_1 + \lambda_2) & 0 & \theta_1^2(\lambda_1 + \lambda_2) & \lambda_2^2 \end{bmatrix} \quad (D2-5b)$$

or

$$\bar{A}^{-2} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 9 & -1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ -6 & 0 & -3 & -1 \end{bmatrix} - \begin{bmatrix} 2 & 0 & 0 & 0 \\ 9 & -1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ -6 & 0 & -3 & -1 \end{bmatrix} \quad (D2-6a)$$

$$= \begin{bmatrix} 4 & 0 & 0 & 0 \\ 9 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ -6 & 0 & 9 & 1 \end{bmatrix} \quad (D2-6b)$$

and

$$\bar{A}^{-3} = \left[\begin{array}{cc|cc} \lambda_1^2 & 0 & 0 & 0 \\ \theta_1^2(\lambda_1 + \lambda_2) & \lambda_2^2 & 0 & 0 \\ \hline 0 & 0 & \lambda_1^2 & 0 \\ 2\theta_1(\lambda_1 + \lambda_2) & 0 & \theta_1^2(\lambda_1 + \lambda_2) & \lambda_2^2 \end{array} \right] \quad \left[\begin{array}{cc|cc} \lambda_1 & 0 & 0 & 0 \\ \theta_1^2 & \lambda_2 & 0 & 0 \\ \hline 0 & 0 & \lambda & 0 \\ 2\theta_1 & 0 & \theta_1^2 & \lambda_2 \end{array} \right] \quad (D2-7a)$$

$$= \left[\begin{array}{cc|cc} \lambda_1^3 & 0 & 0 & 0 \\ \theta_1^2(\lambda_1^2 + \lambda_1\lambda_2 + \lambda_2^2) & \lambda_2^3 & 0 & 0 \\ \hline 0 & 0 & \lambda_1^2 & 0 \\ 2\theta_1(\lambda_1^2 + \lambda_1\lambda_2 + \lambda_2^2) & 0 & \theta_1^2(\lambda_1^2 + \lambda_1\lambda_2 + \lambda_2^2) & \lambda_2^3 \end{array} \right] \quad (D2-7b)$$

or

$$\bar{A}^{-3} = \left[\begin{array}{cc|cc} 4 & 0 & 0 & 0 \\ 9 & 1 & 0 & 0 \\ \hline 0 & 0 & 4 & 0 \\ -6 & 0 & 9 & 1 \end{array} \right] \quad \left[\begin{array}{cc|cc} 2 & 0 & 0 & 0 \\ 9 & -1 & 0 & 0 \\ \hline 0 & 0 & 2 & 0 \\ -6 & 0 & -3 & -1 \end{array} \right] \quad (D2-8a)$$

$$= \left[\begin{array}{cc|cc} 8 & 0 & 0 & 0 \\ 27 & -1 & 0 & 0 \\ \hline 0 & 0 & 8 & 0 \\ -18 & 0 & 27 & -1 \end{array} \right] \quad (D2-8b)$$

Notice from Eqs (D2-2, D2-5, D2-7) that the parameter and its sensitivity remain in their respective positions for all the powers of the augmented matrix. This is an important property because it causes all the modes to remain visible throughout the component matrix process of system identification.

State Transition Matrix. For this example, Eq (C2-8) is

$$e^{\bar{A}t} = a_1(t)I + a_2(t)\bar{A} + a_3(t)\bar{A}^2 + a_4(t)\bar{A}^3 \quad (D2-9)$$

Eq (C2-7) results in the following system of linear equations, from which the remainder polynomial coefficients, $a_j(t)$, may be found:

$$\begin{bmatrix} a_1(t) \\ a_2(t) \\ a_3(t) \\ a_4(t) \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ v_{41} & v_{42} & v_{43} & v_{44} \end{bmatrix} \begin{bmatrix} e^{\lambda_1 t} \\ \lambda_1 t \\ te^{\lambda_1 t} \\ te^{\lambda_1 t} \end{bmatrix} \quad (D2-10)$$

By applying Eq (D2-4), the following coefficients are found:

$$\begin{aligned} a_1(t) &= 21/81e^{2t} - 18/81te^{2t} + 60/81e^{-t} + 36/81te^{-t} \\ a_2(t) &= 36/81e^{2t} - 27/81te^{2t} - 36/81e^{-t} \\ a_3(t) &= 9/81e^{2t} - 9/81e^{-t} - 27/81te^{-t} \\ a_4(t) &= -6/81e^{2t} + 9/81te^{2t} + 6/81e^{-t} + 9/81te^{-t} \end{aligned} \quad (D2-11)$$

Substituting into Eq (D3-9):

$$e^{\bar{A}t} = (21/81e^{2t} - 18/81te^{2t} + 60/81e^{-t} + 36/81te^{-t}) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$+ (36/81e^{2t} - 27/81te^{2t} - 36/81e^{-t})$$

$$\left[\begin{array}{cc|cc} 2 & 0 & 0 & 0 \\ \theta_1^2 & -1 & 0 & 0 \\ \hline 0 & 0 & 2 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & -1 \end{array} \right]$$

$$+ (9/81e^{2t}$$

$$-9/81e^{-t} - 27/81te^{-t})$$

$$\left[\begin{array}{cc|cc} 4 & 0 & 0 & 0 \\ \theta_1^2 & 1 & 0 & 0 \\ \hline 0 & 0 & 4 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & 1 \end{array} \right]$$

$$+ (-6/81e^{2t} + 9/81te^{2t} + 6/81e^{-t} + 9/81te^{-t})$$

$$\left[\begin{array}{cc|cc} 8 & 0 & 0 & 0 \\ 3\theta_1^2 & -1 & 0 & 0 \\ \hline 0 & 0 & 8 & 0 \\ 6\theta_1 & 0 & 3\theta_1^2 & -1 \end{array} \right]$$

(D2-12)

Summing up all the terms yields

$$\bar{A}t = \left[\begin{array}{cc|cc} e^{2t} & 0 & 0 & 0 \\ \theta_1^2(27/81e^{2t} - 27/81e^{-t}) & e^{-t} & 0 & 0 \\ \hline 0 & 0 & e^{2t} & 0 \\ \theta_1(54/81e^{2t} - 54/81e^{-t}) & 0 & \theta_1^2(27/81e^{2t} - 27/81e^{-t}) & e^{-t} \end{array} \right]$$

(D2-13)

If Eq (D2-13) is evaluated at $\theta_1 = -3$, the result is

$$e^{\bar{A}t} = \left[\begin{array}{cc|cc} e^{2t} & 0 & 0 & 0 \\ (3e^{2t} - 3e^{-t}) & e^{-t} & 0 & 0 \\ \hline 0 & 0 & e^{2t} & 0 \\ (-2e^{2t} + 2e^{-t}) & 0 & (3e^{2t} - 3e^{-t}) & e^{-t} \end{array} \right] \quad (D2-14)$$

Parameter Sensitivity Derivatives. This system has only one parameter sensitivity derivative, $e_{(1)}^{At}$, and it can be found by applying Eq (19), which for this example is

$$e_{(1)}^{At} = \sum_{j=1}^{2n=4} A_{(1)}^{j-1} a_j(t) \quad (D2-15)$$

From Eq (D2-15) it is evident that powers of the original system matrix are needed up through the third power. Therefore

$$A^2 = \begin{bmatrix} \lambda_1 & 0 \\ \theta_1^2 & \lambda_2 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ \theta & \lambda_2 \end{bmatrix} = \begin{bmatrix} \lambda_1^2 & 0 \\ \theta_1^2(\lambda_1 + \lambda_2) & \lambda_2^2 \end{bmatrix} \quad (D2-16)$$

or

$$A^2 = \begin{bmatrix} 2 & 0 \\ 9 & -1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 9 & -1 \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 9 & 1 \end{bmatrix} \quad (D2-17)$$

and

$$\begin{aligned} A^3 &= \begin{bmatrix} \lambda_1^2 & 0 \\ \theta_1^2(\lambda_1 + \lambda_2) & \lambda_2^2 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ \theta_1^2 & \lambda_2 \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1^3 & 0 \\ \theta_1^2(\lambda_1^2 + \lambda_1\lambda_2 + \lambda_2^2) & \lambda_2^3 \end{bmatrix} \end{aligned} \quad (D2-18)$$

or

$$A^3 = \begin{bmatrix} 4 & 0 \\ 9 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 9 & -1 \end{bmatrix} = \begin{bmatrix} 8 & 0 \\ 27 & -1 \end{bmatrix} \quad (D2-19)$$

Applying these to Eq (D2-15):

$$\begin{aligned}
 e_{(1)}^{At} = & \frac{d}{d\theta_1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} (2/81e^{2t} - 18/81te^{2t} + 60/81e^{-t} + 36/81te^{-t}) \\
 & + \frac{d}{d\theta_1} \begin{bmatrix} 2 & 0 \\ \theta_1^2 & -1 \end{bmatrix} (36/81e^{2t} - 27/81te^{2t} - 36/81e^{-t}) \\
 & + \frac{d}{d\theta_1} \begin{bmatrix} 4 & 0 \\ \theta_1^2 & 1 \end{bmatrix} (9/81e^{2t} - 9/81e^{-t} - 27/81te^{-t}) \\
 & + \frac{d}{d\theta_1} \begin{bmatrix} 8 & 0 \\ 3\theta_1^2 & -1 \end{bmatrix} (-6/81e^{2t} + 9/81te^{2t} + 6/81e^{-t} + 9/81te^{-t})
 \end{aligned}$$

(D2-20a)

$$\begin{aligned}
 = & \begin{bmatrix} 0 & 0 \\ 2\theta_1 & 0 \end{bmatrix} (36/81e^{2t} + 27/81te^{2t} - 36/81e^{-t}) \\
 & + \begin{bmatrix} 0 & 0 \\ 2\theta_1 & 0 \end{bmatrix} (9/81e^{2t} - 9/81e^{-t} - 27/81te^{-t}) \\
 & + \begin{bmatrix} 0 & 0 \\ 6\theta_1 & 0 \end{bmatrix} (-6/81e^{2t} + 9/81te^{2t} + 6/81e^{-t} + 9/81te^{-t})
 \end{aligned}$$

(D2-20b)

Finally,

$$e_{(1)}^{At} = \begin{bmatrix} 0 & 0 \\ \theta_1 (54/81e^{2t} - 54/81e^{-t}) & 0 \end{bmatrix}$$

(D2-21)

A comparison of Eq (D2-21) and Eq (D2-13) shows that the parameter sensitivity of the original state transition matrix is in fact the lower left block of the partitioned state transition matrix for the augmented (sensitivity) system. Thus the validity of Eq (15) is demonstrated.

Use of Component Matrices. Since the augmented system has two distinct modes and two pseudo modes, a total of four component matrices must be found: $Z_{10}, Z_{11}, Z_{20}, Z_{21}$. The augmented system state transition matrix and the component matrices may be found using Eqs (C3-2) and (C3-1), respectively. They are repeated here as Eqs (D2-22) and (D2-23).

$$e^{\bar{A}t} = \sum_{j=1}^M \sum_{k=0}^{m_j-1} (Z_{j,k}) t^k e^{\lambda_j t} \quad (D2-22)$$

$$Z_{j,k} = \sum_{\ell=1}^n v_{\ell,c} \bar{A}^{\ell-1}; c = \begin{bmatrix} j \\ \sum_{i=1}^j m_i \end{bmatrix} - m_j + k + 1 \quad (D2-23)$$

Applying previous calculations:

$$Z_{10} = \sum_{\ell=1}^{n=4} v_{\ell,1} \bar{A}^{\ell-1} \quad (D2-25a)$$

$$= 21/81 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + 36/81 \begin{bmatrix} 2 & 0 & 0 & 0 \\ \theta_1^2 & -1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & -1 \end{bmatrix}$$

$$+ 9/81 \begin{bmatrix} 4 & 0 & 0 & 0 \\ \theta_1^2 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & 1 \end{bmatrix} - 6/81 \begin{bmatrix} 8 & 0 & 0 & 0 \\ 3\theta_1^2 & -1 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 6\theta_1 & 0 & 3\theta_1^2 & -1 \end{bmatrix}$$

$$= \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ \frac{27}{81}\theta_1^2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{54}{81}\theta_1^2 & 0 & \frac{27}{81}\theta_1^2 & 0 \end{array} \right] \quad (D2-25c)$$

$$z_{11} = \sum_{\ell=1}^4 v_{\ell,2} \bar{A}^{\ell-1} \quad (D2-26a)$$

$$= -18/81 \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] - 27/81 \left[\begin{array}{cc|cc} 2 & 0 & 0 & 0 \\ \theta_1^2 & -1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & -1 \end{array} \right]$$

$$+ 9/81 \left[\begin{array}{cc|cc} 8 & 0 & 0 & 0 \\ 3\theta_1^2 & -1 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 6\theta_1 & 0 & 3\theta_1^2 & -1 \end{array} \right] \quad (D2-26b)$$

$$= \left[\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right] \quad (D2-26c)$$

$$z_{20} = \sum_{\ell=1}^4 v_{\ell,3} \bar{A}^{\ell-1} \quad (D2-27a)$$

$$= 60/81 \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] - 36/81 \left[\begin{array}{cc|cc} 2 & 0 & 0 & 0 \\ \theta_1^2 & -1 & 0 & 0 \\ \hline 0 & 0 & 2 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & -1 \end{array} \right]$$

$$- 9/81 \left[\begin{array}{cc|cc} 4 & 0 & 0 & 0 \\ \theta_1^2 & 1 & 0 & 0 \\ \hline 0 & 0 & 4 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & 1 \end{array} \right] + 6/81 \left[\begin{array}{cc|cc} 8 & 0 & 0 & 0 \\ 3\theta_1^2 & -1 & 0 & 0 \\ \hline 0 & 0 & 8 & 0 \\ 6\theta_1 & 0 & 3\theta_1^2 & -1 \end{array} \right]$$

(D2-27b)

$$= \left[\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ -\frac{27}{81}\theta_1^2 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ -\frac{54}{81}\theta_1 & 0 & -\frac{27}{81}\theta_1^2 & 1 \end{array} \right]$$

(D2-27c)

$$z_{21} = \sum_{\ell=1}^4 v_{\ell,4} \bar{A}^{\ell-1}$$

(D2-28a)

$$= 36/81 \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] - 27/81 \left[\begin{array}{cc|cc} 4 & 0 & 0 & 0 \\ \theta_1^2 & 1 & 0 & 0 \\ \hline 0 & 0 & 4 & 0 \\ 2\theta_1 & 0 & \theta_1^2 & 1 \end{array} \right]$$

$$+ 9/81 \left[\begin{array}{cc|cc} 8 & 0 & 0 & 0 \\ 3\theta_1^2 & -1 & 0 & 0 \\ \hline 0 & 0 & 8 & 0 \\ 6\theta_1 & 0 & 3\theta_1^2 & -1 \end{array} \right]$$

(D2-28b)

$$= \left[\begin{array}{cc|cc} a & a & 0 & a \\ 0 & 0 & 0 & 0 \\ \hline a & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right] \quad (D2-28c)$$

It can be seen from the above that if the component matrices are calculated in order of increasing value of the subscript, the calculation of c , the column index for $v_{l,c}$, need not be computed. It can be simply initialized to one for the first component matrix, and thereafter incremented by one for each new component matrix calculation.

Substituting the component matrices into Eq (D2-22) yields the state transition matrix for the augmented system:

$$e^{\bar{A}t} = \sum_{j=1}^{M=2} \sum_{k=0}^{m_j-1} (z_{j,k} t^k e^{\lambda_j t} \quad (D2-29a)$$

$$= z_{10} e^{\lambda_1 t} + z_{11} t e^{\lambda_1 t} + z_{20} e^{\lambda_2 t} + z_{21} t e^{\lambda_2 t} \quad (D2-29b)$$

$$= \left[\begin{array}{cc|cc} 1 & a & a & 0 \\ \frac{27}{81} \theta_1^2 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \frac{54}{81} \theta_1 & 0 & \frac{27}{81} \theta_1^2 & a \end{array} \right] e^{2t} + [0] t e^{2t} \\ + \left[\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ -\frac{27}{81} \theta_1^2 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ -\frac{54}{81} \theta_1 & 0 & -\frac{27}{81} \theta_1^2 & 1 \end{array} \right] e^{-t} + [0] t e^{-t} \quad (D2-29c)$$

$$= \left[\begin{array}{cc|cc} e^{2t} & 0 & 0 & 0 \\ \theta_1^2 (27/81e^{2t} - 27/81e^{-t}) & e^{-t} & 0 & 0 \\ \hline 0 & 0 & e^{2t} & 0 \\ \theta_1 (54/81e^{2t} - 54/81e^{-t}) & 0 & \theta_1^2 (27/81e^{2t} - 27/81e^{-t}) & e^{-t} \end{array} \right] \quad (D2-29d)$$

Comparing Eq (D2-29d) with Eq (D2-13) shows that the same solution is found by either method. Again the influence of the system modes and parameters is clearly visible throughout the component matrix method solution process.

Example 3 - Use of Csáki's Generalized Algorithm

An explanation of Csáki's generalized algorithm is found in Section III of the main text. Suppose for this example that a fourth order system has one distinct eigenvalue and one eigenvalue repeated three times: $\lambda_1 = 1$ with multiplicity $k_1 = 3$, and $\lambda_2 = -1$ with multiplicity $k_2 = 1$. The Vandermonde matrix can then be written

$$V = \left[\begin{array}{cccc} 1 & 0 & 0 & 1 \\ \lambda_1 & 1 & 0 & \lambda_2 \\ \lambda_1^2 & 2\lambda_1 & 1/2(2) & \lambda_2^2 \\ \lambda_1^3 & 3\lambda_1^2 & 1/2(6\lambda_1) & \lambda_2^3 \end{array} \right] = \left[\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & -1 \\ 1 & 2 & 1 & 1 \\ 1 & 3 & 3 & -1 \end{array} \right] \quad (D3-1)$$

which has an inverse

$$V^{-1} = \begin{bmatrix} 7/8 & 3/8 & -3/8 & 1/8 \\ -6/8 & 2/8 & 6/8 & -2/8 \\ 4/8 & -4/8 & -4/8 & 4/8 \\ 1/8 & -3/8 & 3/8 & -1/8 \end{bmatrix} \quad (D3-2)$$

The characteristic polynomial is

$$D(S) = (S-1)^3(S+1) = S^4 - 2S^3 + 0S^2 + 2S - 1 \quad (D3-3)$$

There are two denominator polynomials--one for each eigenvalue. They are

$$D_1(S) = \frac{D(S)}{(S-\lambda_1)^{k_1}} = \frac{S^4 - 2S^3 + 0S^2 + 2S - 1}{(S-1)^3} = S + 1 \quad (D3-4)$$

$$D_2(S) = \frac{D(S)}{(S-\lambda_2)^{k_2}} = \frac{S^4 - 2S^3 + 0S^2 + 2S - 1}{(S+1)} = (S-1)^3 = S^3 - 3S^2 + 3S - 1 \quad (D3-5)$$

For every application of Csáki's algorithm there are n truncated polynomials, $N_\ell(S)$. In this example they are:

$$N_1(S) = S^3 - 2S^2 + 0S + 2 \quad (D3-6)$$

$$N_2(S) = S^2 - 2S = 0 \quad (D3-7)$$

$$N_3(S) = S - 2 \quad (D3-8)$$

$$N_4(S) = 1 \quad (D3-9)$$

The following structure describes the inverse Vandermonde for this example:

$$V^{-1} = \begin{bmatrix} W_1 \\ \vdots \\ W_2 \end{bmatrix} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} & W_{33}^{(1)} & W_{34}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ W_{11}^{(2)} & W_{12}^{(2)} & W_{13}^{(2)} & W_{14}^{(2)} \end{bmatrix} \quad (D3-10)$$

Each element is evaluated as follows, using Eq (25), repeated here as Eq (D3-11):

$$W_{jl}^{(i)} = \frac{1}{(k_i - j)!} \left. \frac{d^{(k_i - j)}}{ds^{(k_i - j)}} \frac{N_l(s)}{D_i(s)} \right|_{s=\lambda_1} \quad (D3-11)$$

$$W_{11}^{(1)} = \frac{1}{(3-1)!} \left. \frac{d^2}{ds^2} \frac{N_1(s)}{D_1(s)} \right|_{s=\lambda_1=1} = \frac{1}{2} \left. \frac{d^2}{ds^2} \frac{s^3 - 2s^2 + 2}{s+1} \right|_{s=1} = \frac{7}{8} \quad (D3-12a)$$

$$W_{12}^{(1)} = \frac{1}{(3-1)!} \left. \frac{d^2}{ds^2} \frac{N_2(s)}{D_1(s)} \right|_{s=\lambda_1=1} = \frac{1}{2} \left. \frac{d^2}{ds^2} \frac{s^2 - 2s}{s+1} \right|_{s=1} = \frac{3}{8} \quad (D3-12b)$$

$$W_{13}^{(1)} = \frac{1}{(3-1)!} \left. \frac{d^2}{ds^2} \frac{N_3(s)}{D_1(s)} \right|_{s=\lambda_1=1} = \frac{1}{2} \left. \frac{d^2}{ds^2} \frac{s-2}{s+1} \right|_{s=1} = \frac{3}{8} \quad (D3-12c)$$

$$W_{14}^{(1)} = \frac{1}{(3-1)!} \left. \frac{d^2}{ds^2} \frac{N_4(s)}{D_1(s)} \right|_{s=\lambda_1=1} = \frac{1}{2} \left. \frac{d^2}{ds^2} \frac{1}{s+1} \right|_{s=1} = \frac{1}{8} \quad (D3-12d)$$

etc.

After evaluating each element, the matrix formed from Eqs (D3-10) and (D3-11) is the same as the matrix, Eq (D3-2), found by inverting the Vandermonde matrix. Note, however, that Csáki's method does not need to work with the Vandermonde matrix at all. It only needs the system eigenvalues and their multiplicities.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|--|-----------------------|--|
| 1. REPORT NUMBER AFIT/GGC/EE/79-2 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) INVESTIGATION OF INVERSE VANDERMONDE MATRIX CALCULATION FOR LINEAR SYSTEM APPLICATIONS | | 5. TYPE OF REPORT & PERIOD COVERED M S Thesis |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Donald P. Seyler 2nd Lieutenant | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Flight Dynamics Lab/FGC Wright-Patterson AFB OH 45433 | | 12. REPORT DATE March 1979 |
| | | 13. NUMBER OF PAGES 188 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | | 15. SECURITY CLASS. (of this report) Unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |
| 16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited | | |
| 17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) | | |
| 18. SUPPLEMENTARY NOTES Approved for public release; IAW AFR 190-17 Joseph P. Hipps, Major, USAF Director of Information 16 MAY 1979 | | |
| 19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Linear Systems Vandermonde Matrix Inversion Sensitivity Variables Sensitivity Parameters System Identification Matrix Exponential | | |
| 20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Efficient inversion of a $2n \times 2n$ Vandermonde matrix is a key requirement of a new algorithm developed by J. Gary Reid (Ref 10,11) for parameter identification in linear time-invariant systems. It has features very desirable for application to large systems with many unknown parameters, such as adaptive flight control systems. A generalized algorithm for inverting the Vandermonde matrix was proposed by F. G. Csaki (Ref 1). It was chosen for study because its structure parallels (over) | | |

DD FORM 1473

EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

that of ^{the first} ~~Reid's~~ algorithm. ^{The} ~~esaki's~~ algorithm was coded into a computer sub-routine called ~~YANINY~~, and 43 eigensystems were used to test its computational accuracy and efficiency. Four potential problem areas tested were: (1) large system orders, (2) eigenvalues near each other, (3) eigenvalues near zero, and (4) eigenvalues with large magnitude differences. For a comparison, the Vandermonde matrix for each system was also inverted using routines from the International Mathematical & Statistical Library (IMSL).

Test results indicated that VANINV is not quite as fast or as accurate as the IMSL routines. However, the tests were limited to systems with real distinct eigenvalues because the IMSL routines cannot handle other types. VANINV in its present form can handle systems having any combination of complex and/or repeated eigenvalues. Therefore, recommendations for further research and several possible means of improving VANINV are outlined.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)